2020

AUTOMATED GENERATION OF DETAILED PROGRAMMING ASSIGNMENT FEEDBACK

Daniel L. Gauthier
University of Rhode Island, dangauthier8@gmail.com

AUTOMATED GENERATION OF DETAILED PROGRAMMING ASSIGNMENT FEEDBACK

BY

DANIEL L. GAUTHIER

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

2020

MASTER OF SCIENCE THESIS

OF

DANIEL L. GAUTHIER

APPROVED:

Thesis Committee:

Major Professor Victor Fay-Wolfe

Lisa DiPippo

Jay Fogleman Jr

Nasser H. Zawia
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2020

ABSTRACT

When teaching students computer programming, instructors often teach specific techniques that students should follow. Students are told to program in these ways, but instructors never really know if the techniques are used; and if they are used, then how effective they are. This project produced a Programming Analysis Plug-In (PAPI) to analyze student academic computer programming course work to measure when and how students are working on programming assignments.These measurements include examining the final assignment submitted by a student as well as the steps a student used to get to the final product. To make sure that this data capture is being performed in the most user-friendly way, potential users, both instructors and students, were interviewed for their opinions on how the software should work. It was determined both students and instructors prefer auto-grading software, but it currently lacks formative feedback. It was also postulated that if a teacher can access and easily understand how a student gets to a final result, they can help support struggling students, find class pain points, and discover bad practices on projects. Having an instructor sit down with every student to ask how they programmed an assignment is not feasible in the large classes found in computer science. By automating this process, students can get the feedback that they need to excel. PAPI delivers this feedback by analyzing assignment creation date, last edit date, number of saves, number of character insertions and deletions, and number of comments. This thesis describes the PAPI software, its testing in a computer science course, and the results that indicated starting an assignment early, commenting code, and having lower numbers of text insertions and deletions trend to higher assignment grades. PAPI will have a broad impact because of its compatibility with current technologies and its intuitive ease of use.

ACKNOWLEDGMENTS

I would like to express my sincerest gratitude towards my major professor and thesis advisor, Dr. Victor Fay-Wolfe, for his time, guidance, and support throughout this research project. In a time of uncertainty due to the COVID-19 pandemic, his willingness to help allowed me to maintain a proper timeline while ensuring I met my goals. I cannot thank him enough for his encouragement to pursue my Master's degree in Computer Science. The opportunities he has provided me have changed my academic career and empowered me to flourish to a point I had not previously thought possible.

I would also like to thank my committee members, Dr. Lisa DiPippo and Dr. Jay Fogleman Junior, for always being just an email away. Their commitment to my success and thoughtful recommendations have been invaluable in my educational journey. It is because of their instruction that I have been able to pinpoint challenging problems before they arise.

I would like to thank Jake Fonseca for his guidance in the early stages of this research project. His mentorship gave me a thought-provoking forum to discuss thesis topics and find one I am truly interested in. Beyond this, I was able to explore a topic with real-world implications that can directly aid others upon deployment. Over the course of my schooling, his direction has allowed me to own my education and understand what I want and what I need from my degree, all while enjoying the process.

I feel a deep sense of gratitude towards my parents, sister, and boyfriend for their persistent encouragement, love, and words of wisdom. They give me the confidence and determination I need to succeed beyond just the constraints of the classroom.

I thank these people for their backing; they have made this research possible.

TABLE OF CONTENTS

Page

Page

Page

LIST OF FIGURES

Figure **Page**

Figure **Page**

LIST OF TABLES

CHAPTER 1

Introduction

1.1 Background

When teaching students new programming skills, an instructor must have various ways to evaluate the student. A student's progress throughout a class must be recorded and evaluated in order for the student to know where they need to improve. In typical computer science courses, students hand assignments in without the instructor knowing how they got to the end product. If student progress on an assignment can be recorded and analyzed, then formative feedback can be provided to the students and instructors.

With the capability to access and understand how a student progressed to their final result, the teacher can then help support struggling students, find class-wide patterns, and discover discouraged programming practices on projects. Traditionally, this data can only be obtained by having an instructor sit down with every student to ask how they coded a program. This is unrealistic for large class sizes with limited staff instruction time. Class sizes are growing, and at some universities, they are expected to reach a capacity where the instructor cannot grade every assignment manually[1]. Coding is becoming an important and necessary skill for employability and the current numbers of computer science students will only increase [2]. Ways to grade assignments to provide summative feedback via auto-grading software exist. However, with these current auto-grading tools, students do not receive the traditional formative feedback that helps improve their grades [3]. Previous research has been done in the areas of proper programming techniques and the efficiency of auto-grading assignments. If this formative feedback can be provided in an automated fashion, like summative grading is done now, this support can be included while not increasing the workload of the teaching staff.

In this project, I followed the standard techniques used in software development - I researched the needs and wants of potential users of a tool that records student approaches to programming and built to those requirements. This process included having one-on-one interviews with the target users.

1.2 Motivation

In my computer science undergraduate career, I noticed an increase in auto-grading assignments. While I saw many of the benefits to this technology, I felt inhibited by its shortcomings. In particular, I wanted more feedback on how I work as a programmer and if there were specific patterns in my style that could be improved. From this clear need in the computer science education process that I experienced, I decided that I would base my project on the analysis of students' programming process to identify patterns that could be useful for instructors.

1.3 Research Goals

In this project, research and development was performed to establish the implementation of an integrated development environment (IDE) plugin that is able to generate auto-graded formative feedback. The research focused on four main points:

- Find what data can be and should be collected from a student assignment;
- Development of software to generate/parse the data;
- Find patterns in the student data;
- Provide the ability to allow others to analyze their own data captured from the tool.

The goals of this work are explained in more detail in the following subsections.

1.3.1 Formative Feedback

My project included developing software to analyze student work to automatically provide formative feedback to both the student and to instructors, along with a plagiarism indicator. The purpose of PAPI is to identify patterns in a student's programming technique. This feedback can be delivered to the professor, who can choose to do things such as help students struggling on assignments, use it at the end of the semester for the cumulative patterns of students, or see how different teaching styles affect student work. The feedback software also generates a PDF report that can be sent to the student to reflect on their own patterns. When analyzing a student's assignment, PAPI can help identify plagiarized work by checking the amount of entered text in a specific amount of time. The software will not only detect plagiarized work, but all copy and pasted text. PAPI checks for this by only looking at the keystroke information provided by the student. Any text, including the students own written code, will be flagged if inserted at an accelerated speed. The analysis of this tool, presented in Chapter 4, shows that copied (assumed to be cheated) code is identified with at least 80% accuracy with no more than 10% false positives and 10% false negatives.

1.3.2 Workload

Another important criteria is the amount of extra effort that students and instructors have to perform to use a feedback tool. After prototyping the tool, I found that it requires minimal effort by the teaching staff. PAPI can generate a student's report in less than seven clicks by a user. The feedback time for generating a report is also immediate, at a rate of less than one second a student.

1.3.3 Wide Impact

PAPI has a wide impact on computer science education by ensuring integration with already existing technologies. PAPI functions for both the University of Rhode Island, as well as any additional environment where the same technologies are already being used. PAPI should be able to reach at least 500 students at the University of Rhode Island and at least 10,000 nationally. This is done by having PAPI seamlessly integrate with current tools used in teaching computer programming.

List of References

[1] A. Nguyen, C. Piech, J. Huang, and L. Guibas, "Codewebs: Scalable homework search for massive open online programming courses," in *Proceedings of the 23rd International World Wide Web Conference (WWW 2014)*, Seoul, Korea, 2014.

[2] M. J. Handel, "Trends in job skill demands in oecd countries," no. 143, 2012. [Online]. Available: https://www.oecd-ilibrary.org/content/paper/5k8zk8pcq6td-en

[3] A. Vihavainen, M. Paksula, and M. Luukkainen, "Extreme apprenticeship method in teaching programming for beginners," in *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 93–98. [Online]. Available: https://doi.org/10.1145/1953163.1953196

CHAPTER 2

Related Literature

2.1 Online Teaching

Online teaching is traditionally done using a Learning Management System (LMS) to provide a medium for communication between the instructor and the students, and allow for the submission of assignments. Assignments can be graded and returned to students in the LMS as well.

2.1.1 Digital Grading

Classrooms across disciplines are increasing the use of online feedback. Singh et al. looked into if this online feedback matches the quality of the traditional feedback methods. Their discovery shows that, while there are complaints with online grading, a large number of instructors are still using it [1]. Many of the complaints have to do with the online nature of software grading. Issues listed by the instructors included lacking the knowledge of how to use the tool, and not having the same emphasis abilities as traditional grading. However, the use of automated grading is still growing. This has been shown by constant increase in the use of automatic grading tools, as well as one-on-one interviews performed in the study. Speed and efficiency are rated the most important by the surveyed instructors.

2.1.2 Online Only Courses

Massive Open Online Courses, commonly shortened to the acronym *MOOC*, are gaining in popularity in higher education [2]. These courses are overcoming boundaries such as language and cultures through the use of the online setting. The scaling of grading infrastructure is also increasing. When students take an online course, they need to receive their feedback in a faster manner so they can

progress in their curriculum. To keep up with this demand, new ways of grading will need to take place.

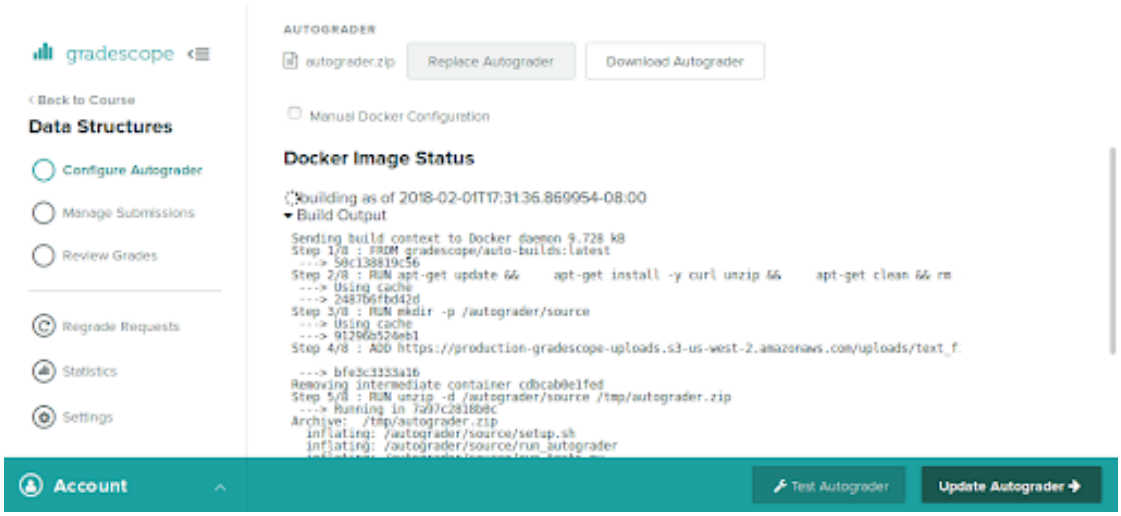


Figure 1. This is a screenshot of the gradescope cloud auto-grading software. Gradescope is a common LMS used for digital grading and automatic grading.

2.2 Auto-graders

Automatically graded assignments in the computer science classroom are becoming increasingly common at all stages of the learning process. Currently, feedback given by auto-grading software provides a grade but does not give specific feedback on the nuances of programming. This feedback includes, but is not limited to, analyzing when the student starts working on an assignment, how much and when the student comments their code, and how the student divides the work of a large assignment[3, 4, 5]. To better learn the skill of programming through personalized feedback, more detail needs to be supplied to the teacher for proper observation. For a student to become a master of their skill faster and to provide a more enjoyable programming experience, a teacher needs more information from the IDE to properly assess and assist the student[6, 7]. The way an auto-grader currently grades student work is limited to the strategies explained below.

2.2.1 Algorithmic Strategy
Teacher Provided Responses

Assignments can be written in many ways. When working on a programming assignment, especially with introductory-level material, there are only a few options to implement an idea[8]. Current implementations for project grading with higher levels of feedback involve a process of creating an algorithm that software can check across the students' work. MISTAKEBROWSER[9] is a current deployment of this style of software. While this level of grading is very helpful for the students, it does not alleviate work from the instructor. The process of writing every possible option to solve an assignment is arduous and repetitive. There is also the issue of not thinking of all possibilities and still having to go back and manually grade those missed students. The purpose of our software is to provide feedback on an assignment without prior knowledge of its internals.

Previous Submission Analysis

With the growth of artificial intelligence (AI) and neural networks, the work can shift from the teacher providing answers to the student. When looking at student submissions of previous semesters, variations exist, but only to a point [10]. The research by Huang et al. looked at 32,876 submissions and found 423 correct ways to solve the problem with close to 3,000 incorrect ways. This supports the idea that the manual teacher generation of these approaches would not reach a level of significance to support the work. It also shows that with a large enough data set, patterns do arise in student programming techniques. Grading future work based on the past works well, but if an assignment is changed at all from semester to the next, the current technology cannot be re-branded for different assignments.

2.2.2 Grammars

A similar approach to an algorithmic strategy is to check a student's assignment at the level of grammars [11]. Instead of providing all options and having set responses, this program looks through the student's assignment with the use of a syntax tree, catching both programming style *and* structure. This works to the teacher's advantage when teaching very specific skills for programming constructs such as logic, loops, and functions. While compiling software can arguably also provide debugging information, teacher-provided comments can exceed this level of detail and explain to the student using beginner-level language and terminology.

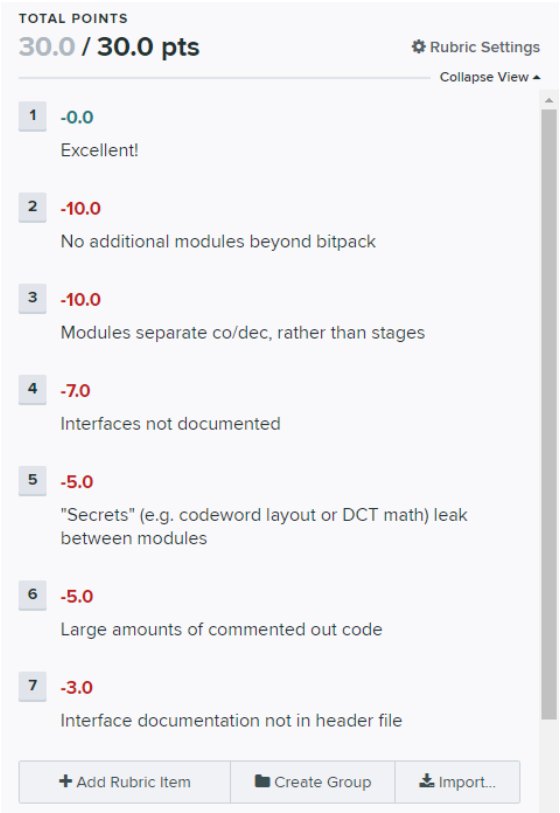


Figure 2. This is a screenshot of possible feedback options for applying previous responses to a current assignment.

2.2.3 Reuse of Feedback

The possibility of re-using feedback from previous iterations of a class is another way teachers are increasing the speed of their grading [9]. Similar to having only a limited number of correct answers to a question, there are also only so many ways a student may do an assignment incorrectly [2]. Writing a detailed response once for a student on a common mistake can increase the level of feedback for all other students who make it in the future. This can be used in combination with the work in AI, as explained previously. By grouping similar students' work before providing feedback, an instructor can find all students who made the same mistake at once. Glassman et al. [12] handle this process by producing a stack of variables; comparing students' similarities of actions to those variables. This normalizes the solutions to the important base elements to both group together similar implementations and simplify the readability of student code. The teacher can then apply feedback to multiple students simultaneously who all attempted a part of an assignment in a related way.

2.2.4 Auto-grading Accuracy

Feedback provided by automatic systems can cover many students even in simple deployments. Singh et al. [2] have determined that because of such similarity between student work, software grading of summative feedback can affect the average student even in simple implementations. Their current technique can provide accurate feedback to 64% of over 1000 submitted assignments in under 10 seconds. This shows that this technology when deployed, even in a poor way, can affect a majority of students. The level of detail either needs to be decreased, or the accuracy of the program increased to reach the desired efficiency level. Our study will find that point of efficiency so teachers and students can get the feedback they need with a higher level of reliability.

2.3 Plagiarism

2.3.1 Plagiarism and Learning

When a student copies and pastes code, it constitutes plagiarism. While cheating of all kinds is discouraged and against most university policies, copy and paste has been shown to hinder students' learning capabilities[13] more than other cheating methods. The study determined a student does not spend time thinking about a concept when copy and pasting, leading to a higher rate of forgetting a concept in the future. Students who pasted fewer words were found to have a deeper understanding of material compared to those who pasted more [14]. The processing of material appears to happen less so for students who were not limited in how much text could be pasted. In terms of coding, copy and pasting can be done from websites and other student files. The text can also be copied and pasted from their own files and even in the same document. By not typing out even their own code, they can be losing this memorization[15, 16]. This can lead to what can be referred to as copy-paste driven development or cargo cult programming [17, 18], where a student thinks they understand how a program works, when not really understanding the underlying concepts.

2.3.2 Plagiarism Software

Statistical Analysis

One way to evaluate plagiarism is to look at the programming style using markers and statistical analysis [19] by finding themes in the indentation, variable naming, and spacing. This technique works well, but it assumes the offender is not trying to hide the fact that there is plagiarism occurring. Taking copied code and styling it to the individual's traditional style will remove the opportunity for detecting the difference.

Hashing

Fingerprinting assignments with URL hashing is another way to check for plagiarism incidents [20]. Gao created an algorithm for identifying copies of code between HTML pages. This was created to crawl the web for identical pages to find duplicate websites. While current hashing algorithms, like MD5, work well with detecting exact copies, the fingerprinting method was created to look for similarities. The algorithm was a success and was able to identify many copied pages over the internet. When applying this to web news, duplicated pages ranged from 33.4% to 63.7%. In terms of applying this to the classroom, this would have a direct correlation to classes that teach web-programming. PAPI is also assumed to be modifiable to look over other programming languages.

Winnowing

Measure Of Software Similarity (MOSS) is the most widely used tool at the University of Rhode Island for detecting plagiarism in computer science classes. This software is cloud-based, running most of the process on Stanford owned servers[21]. The service has about 300,000 accounts with 1,000 to 10,000 submissions per day. In each of these submissions, there is a range of twenty to 2,000 assignments [22]. The algorithm of choice for this software is called *winnowing*. MOSS creates small windows of varying sizes to analyze each assignment and assign a hash to locations throughout the provided documents. Once these smaller fingerprints have been made, MOSS looks up each piece of text over the documents. If a fingerprint comes up in two documents, a case of plagiarism is noted and is accumulated to present to the user[23].

Kaya and Özel have taken the MOSS source code plagiarism detection tool [24] and integrated it with the learning management system (LMS) Moodle. By integrating the two together it makes it easier for instructors to check for students

who plagiarize code with the goal of increasing the number of classes that use the technology. In turn, this decreases cheating instances, as the students know they will be caught [25]. Other plagiarism tools are built into the LMS, but none that lend well to the programming classroom. The study found that professors can get plagiarism feedback with little user integration, showing the success of their product. It was also determined that students who have been caught plagiarising tend to have to lower grades in the course. With a success in implementing the auto-grader in the majority of the computer science classrooms it would be assumed there would also be an increase in catching cheating cases. This was not the result. The study determined that due to plagiarism detection software, computer science students became less likely to cheat in fear of getting caught.

JPlag

JPlag is a coding plagiarism detection tool created by Prechelt et al. to analyze Java, Scheme, C, or C++ programs if they contain code similar in multiple assignment submissions [26]. This software is created to build off the previous technology YAP3 [27], but uses new optimizations to improve JPlag 's speed. JPlag hosts a website with separate user accounts to handle queries. A set of assignment submissions are uploaded and are then compared in a pairwise manner. JPlag splits the program into token strings and uses the "Greedy String Tiling" algorithm to find similar code.

Our implementation differs from other plagiarism software as it does not compare the similarity between program structure.

2.4 Programming Techniques

When it is time to start programming there is consensus in the computer science community that there are good and bad ways to write code [28, 29, 30, 31].

The goal of adopting a method of programming is to stay away from the idea of being a "spaghetti code programmer"[30]. This type of programmer throws everything at a program and just sees what sticks. The code is difficult to follow, lacking similarities between sections of the same program. Palomba et. al have created a "smell" detector to sense malpractices like this that can cause problems later on. These technologies already existed, but they wanted to create a better system while analyzing what already exists. By examining change history in combination with the end resulting file, more smells were able to be identified[29] than previous technologies. Change history was the key new addition used to enhance this software's performance. This smell test is a similar idea to our software which will identify students who need help with coursework.

2.4.1 Programming Order of Operations

When teaching and learning how to program, there are many techniques to instruct a student. De Oliveira et al. have identified these moments as a sequence of questions. The programmer has to figure out what to do, how to do it, and then determine what to show[32]. The "what to do" step can be figured out using pseudocode to plan out a program's function. This preparatory step would work off of what an assignment instructs, and starts the process of converting pseudocode to real code. The "how to do" step focuses on the back-end of software to perform operations. Finally, the output of the code is managed and displayed to the user in the "what to show" step.

The importance of commenting code during the programming process helps the software engineer understand their own code and aids in figuring out what the code is doing when others need comments for reference [33, 34]. Currently, this commenting step is frequently skipped over and forgotten. Commenting code is something that students need to start implementing in assignments today as it

is desirable in the software engineering industry [35]. To maintain software with many contributors, this step is even more crucial [36].

Moving onto the "how to do" step, it is important to add logic of the program in a tactful, limited way. While over-commenting can be a problem, overusing logic can affect the readability of code in a more impactful way. If a flow is disrupted frequently, then the code becomes less understandable. This practice can lead to writing a program "to clever"[31]. It takes too much time for someone to understand what a program is doing, and there is little improvement in efficiency. Programming is a team effort and others need to be able to understand what you have written, especially in the classroom setting[37].

Oman and Cook look to find ways to use a taxonomy to talk about different programming styles[38]. In our research, we are looking for which programming methods lead to different grade results. This study acknowledges the importance of programming style but there needs to be improvements in the vocabulary in order to properly have a conversation about the topic.

2.4.2 Programming Method

When trying to apply an "extreme apprenticeship method," a study found that using specific practices provided a better experience for the students participating. An extreme apprenticeship method of teaching, also referenced as *work based learning*, is when education is paired directly with the workplace. A student will learn a concept and then quickly use that skill in a company/real-world setting. Practices such as starting early, having small goals, and having assignments with real-world examples, were elements stressed when holding exercise sessions [28]. The study came to the conclusion that when the practices learned from the mentors are taught to the students along with continuous feedback and great scaffolding lead to higher pass rates for both their introduction to programming and

advanced programming courses. This idea of incremental design appears across computer science disciplines with stressed importance[39].

List of References

[1] B. Price and M. Petre, "Teaching programming through paperless assignments: An empirical evaluation of instructor feedback," in *Proceedings of the 2nd Conference on Integrating Technology into Computer Science Education*, ser. ITiCSE '97. New York, NY, USA: Association for Computing Machinery, 1997, p. 94–99. [Online]. Available: https://doi.org/10.1145/268819.268849

[2] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated feedback generation for introductory programming assignments," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 15–26. [Online]. Available: https://doi.org/10.1145/2491956.2462195

[3] "Gradescope — save time grading." [Online]. Available: https://www.gradescope.com/

[4] "Codegrade - deliver engaging feedback on code." [Online]. Available: https://www.codegrade.com/

[5] "Github classroom." [Online]. Available: https://classroom.github.com/

[6] K. L. Turabian, *A Manual for Writers of Term Papers, Theses, and Dissertations, 6th. edn.* Chicago, Illinois, United States of America: University of Chicago Press, 1987.

[7] University of Rhode Island. "A guide to producing your thesis with latex." June 2006. [Online]. Available: http://www.ele.uri.edu/info/thesis/guide

[8] S. Gulwani, I. Radiček, and F. Zuleger, "Feedback generation for performance problems in introductory programming assignments," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 41–51. [Online]. Available: https://doi.org/10.1145/2635868.2635912

[9] A. Head, E. Glassman, G. Soares, R. Suzuki, L. Figueredo, L. D'Antoni, and B. Hartmann, "Writing reusable code feedback at scale with mixed-initiative program synthesis," in *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, ser. L@S '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 89–98. [Online]. Available: https://doi.org/10.1145/3051457.3051467

[10] J. Huang, C. Piech, A. Nguyen, and L. Guibas, "Syntactic and functional variability of a million code submissions in a machine learning mooc," in *Proceedings of the Workshops at the 16th International Conference on Artificial Intelligence in Education AIED 2013*, 2013.

[11] K. Rivers and K. R. Koedinger, "Automatic generation of programming feedback: A data-driven approach," *AAIED 2013 Workshops Proceedings*, vol. 9, pp. 50–59, 2013.

[12] J. S. Elena L. Glassman, Lyla Fischer and R. C. Miller, "Foobaz: Variable name feedbackfor student code at scale," 2015. [Online]. Available: http://hdl.handle.net/1721.1/116536

[13] K. Monahan, C. Ye, E. Gould, M. Xu, S. Huang, A. Spickard, S. T. Rosenbloom, J. Coco, D. Fabbri, and B. Miller, "Copy-and-paste in medical student notes: Extent, temporal trends, and relationship to scholastic performance," *Applied Clinical Informatics*, vol. 10, no. 3, pp. 479–486, 2019. [Online]. Available: http://dx.doi.org/10.1055/s-0039-1692402

[14] L. B. Igoa and K. A. Kiewra, "How do high-achieving students approach web-based, copy and paste note taking? selective pasting and related learning outcomes," *Journal of Advanced Academics*, vol. 18, no. 4, pp. 512–529, 2007. [Online]. Available: https://journals.sagepub.com/doi/full/10.4219/jaa-2007-558

[15] F. Brack, "Don't copy-paste code. type it out. ?" Jul 2019. [Online]. Available: https://www.freecodecamp.org/news/the-benefits-of-typing-instead-of-copying-54ed734ad849/

[16] M. Rudge, "Why you should never copy and paste code from the web," Mar 2020. [Online]. Available: https://levelup.gitconnected.com/why-you-should-never-copy-and-paste-code-from-the-web-41584544036

[17] C. K. Yuen, "The programmer as navigator: A discourse on program structure," *SIGPLAN Not.*, vol. 18, no. 9, p. 70–78, Sept. 1983. [Online]. Available: https://doi.org/10.1145/988227.988236

[18] H. Thimbleby, "Better programming," 2004.

[19] P. W. Oman and C. R. Cook, "Programming style authorship analysis," in *Proceedings of the 17th Conference on ACM Annual Computer Science Conference*, ser. CSC '89. New York, NY, USA: Association for Computing Machinery, 1989, p. 320–326. [Online]. Available: https://doi.org/10.1145/75427.75469

[20] "The strategy on replicate and similar web collections' detecting and clustering," *Computer applications in engineering education*, vol. 20, pp. 221–231, 2012.

[21] A. Aiken, "A system for detecting software similarity." [Online]. Available: https://theory.stanford.edu/~aiken/moss/

[22] A. Aiken, Jul 2020.

[23] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: Local algorithms for document fingerprinting," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 76–85. [Online]. Available: https://doi.org/10.1145/872757.872770

[24] A. Aiken. Stanford. "A system for detecting software similarity." 2020. [Online]. Available: https://theory.stanford.edu/~aiken/moss/

[25] M. Kaya and S. A. Özel, "Integrating an online compiler and a plagiarism detection tool into the moodle distance education system for easy assessment of programming assignments," *Computer Applications in Engineering Education*, vol. 23, no. 3, pp. 363–373, 2015, iD: $\mathrm{TN}_c di_g ale_i nfotracacademiconefile_A 409574674$.

[26] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with jplag," *Journal of Universal Computer Science*, vol. 8, no. 11, pp. 1016–1038, 2002.

[27] M. J. Wise, "Yap3: Improved detection of similarities in computer program and other texts," in *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 130–134. [Online]. Available: https://doi.org/10.1145/236452.236525

[28] A. Vihavainen, M. Paksula, and M. Luukkainen, "Extreme apprenticeship method in teaching programming for beginners," in *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 93–98. [Online]. Available: https://doi.org/10.1145/1953163.1953196

[29] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk, "Detecting bad smells in source code using change history information," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'13. IEEE Press, 2013, p. 268–278. [Online]. Available: https://doi-org.uri.idm.oclc.org/10.1109/ASE.2013.6693086

[30] S. M. M. Rubiano, O. López-Cruz, and E. G. Soto, "Teaching computer programming: Practices, difficulties and opportunities," in *2015 IEEE Frontiers in Education Conference (FIE)*, 2015, pp. 1–9.

[31] B. W. Kernighan and P. J. Plauger, "Programming style: Examples and counterexamples," *ACM Comput. Surv.*, vol. 6, no. 4, p. 303–319, Dec. 1974. [Online]. Available: https://doi.org/10.1145/356635.356641

[32] M. F. R. B. G. de Oliveira, Clara Amelia; Conte, ""aspects on teaching/learning with object oriented programming for entry level courses of engineering"." ERIC, 1998. [Online]. Available: https://eric.ed.gov/?id=ED448994

[33] P. J. DePasquale, M. E. Locasto, L. Kaczmarczyk, and M. Martinovic, "// todo: Help students improve commenting practices," in *2012 Frontiers in Education Conference Proceedings*, 2012, pp. 1–6.

[34] O. Arafat and D. Riehle, "The commenting practice of open source," in *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, ser. OOPSLA '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 857–864. [Online]. Available: https://doi.org/10.1145/1639950.1640047

[35] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, "A study of the documentation essential to software maintenance," in *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting Designing for Pervasive Information*, ser. SIGDOC '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 68–75. [Online]. Available: https://doi.org/10.1145/1085313.1085331

[36] E. Wong, J. Yang, and L. Tan, "Autocomment: Mining question and answer sites for automatic comment generation," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'13. IEEE Press, 2013, p. 562–567. [Online]. Available: https://doi-org.uri.idm.oclc.org/10.1109/ASE.2013.6693113

[37] K. Stapel, D. Lübke, and E. Knauss, "Best practices in extreme programming course design," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 769–776. [Online]. Available: https://doi.org/10.1145/1368088.1368197

[38] P. W. Oman and C. R. Cook, "A taxonomy for programming style," in *Proceedings of the 1990 ACM Annual Conference on Cooperation*, ser. CSC '90. New York, NY, USA: Association for Computing Machinery, 1990, p. 244–250. [Online]. Available: https://doi.org/10.1145/100348.100385

[39] J. T. Bell, "Extreme programming," 2001.

CHAPTER 3

Design and Methodology

3.1 Student Metrics

Instructors providing formative feedback to students on programming techniques traditionally have required one-on-one sessions for a student to explain how they wrote their code. This takes a lot of time to critique the student's efforts and programming style, and typically does not occur in a vast majority of computer science courses because its time burden is prohibitive. This is even more true for the larger computer science courses that are now common. One of the primary contributions of the PAPI project is to reduce assignment review time drastically. This time reduction was analyzed and compared in this project.

3.1.1 Plagiarism

Plagiarism detection is another feature of PAPI. When a student finds and uses code from the internet or from another student, we assume that the rate of the student keystrokes will be significantly larger compared to other sections of original coding from the student. PAPI's accuracy of identifying copy and pasted code is measured in percent of time correct, false positive, and false negative. PAPI can flag students by identifying the pasted text to help in cheating cases.

3.1.2 Approach Metrics

PAPI works by recording student keystrokes while they programm and helps the instructor and the student identify patterns in the student's programming style. PAPI collects many different forms of metadata and metrics.

One form of metadata to be collected is the **start time of the assignment in relation to the due date**. This data allows an instructor to see when the student started the assignment. For example, if the student started the assignment

when it was first assigned or if they started the day it was due. This could identify and stress the importance of starting an assignment early.

Another form of metadata is the **length of programming sessions**. The amount of time a student puts into a piece of work is assumed to have a correlation to how they succeed. A student who puts less time into an assignment should be flagged to check if effort is being put forth. Programming speed can also be extracted from these measurements for generating a student's total programming time.

The number of programming sessions metadata can help indicate a student's programming method. A student can use this information to identify which work strategy leads to a higher grade on an assignment. They may program many short sessions or a few long sessions. If a large gap of time is found between text insertions, PAPI will count these as two separate work sessions.

Commenting code is traditionally a graded item in early computer science courses, and is metadata that PAPI can capture. Teaching staff view student files to check and grade if the student commented on a program. Automatically searching if a student is commenting and how much they are commenting can relieve an already checked item. PAPI uses regular expressions to scan for comment frequency in the last version of a student file.

The number of saves is also matadata recorded by PAPI to provide more insight into the programming process. A teacher may recommend a student to run and compile their code often. While PAPI cannot check when the code is compiled, the number of save points can be counted and can help indicate which students may not be compiling their code often.

The number of deletions and insertions in terms of characters is metadata that can help indicate when a student is working. If a lot of deletions are

identified this could be a signal that a student struggled on a section. With the University of Rhode Island endorsing the idea of the growth mindset, this struggle and failure was measured to see a possible relationship.

The order in which code elements are written metadata can help indicate when a student is working on different stages of programming. A student may start by writing pseudo-code for an assignment. A student might also not be showing output data until the last step. Both of these can reduce the efficiency of working on a programming assignment. PAPI looks at common strings used at each of these steps to identify where different types of work are being performed.

3.2 Implementation

This section discusses the process of collecting example student work to test the PAPI software.

3.2.1 How to Measure Students Work

My data collection was done in URI's CSC211 course in the Spring of 2020, instructed by Michael Conti. Students do most of their programming in the CS50 IDE in this course. The CS50IDE keeps the history of a student's document progress so the student can go back into save history to undo edits. This is similar to version control found in Microsoft Word, Apple Pages, and Google Docs.

3.2.2 Getting Students History

To make the history accessible the CS50IDE docker image was downloaded and installed to look for this history file, which I found in an SQLite database.

To access the students' files remotely I used a built-in operation referred to as "sharing your work-space". With this action, a user can share their CS50 instance using the cloud version of CS50, authenticated by GitHub.

id	path	contents	fsHash	authAttribs	starRevNums	revNum
Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	lab-01		d41d8cd98f00b204e9800998ecf8427e	[]	[]	0
2	test.cpp		d41d8cd98f00b204e9800998ecf8427e	[]	[]	0
3	projects/file.txt	I love CSC 211	ce59b2c98963ee9ed190eb26b3e3e82c	[14,1]	[1]	1
4	projects/hello.cpp	#include <iostream>...	dceeb5e909ded7fae7dc72a6115349a2	[89,1]	[1]	1
5	lab-02/program1.cpp	#include <iostream>...	d641b5ffc5aa3012624bc6c8e52dfdaa	[278,1]	[27,29,30,32]	32
6	lab-02/program2.cpp	#include <iostream>...	e041924c21e52205bcca05f7defbd75c	[228,1]	[22,23]	26
7	lab-02/program3.cpp	#include <iostream>...	0e825b959f2491b275aa30ad39b56639	[235,null]	[]	0
8	lab-02/test.cpp	#include <iostream>...	9a3d5dcd1c675f6370ed536c4ce310bf	[146,1]	[13,14,15]	15
9	2dash13.cpp	#include <iostream>...	b26990c261add1c81a222838d60ad368	[163,1]	[9,10,11,13,15]	15
10	not_a_lab/2dash13.cpp	#include <iostream>...	b26990c261add1c81a222838d60ad368	[163,null]	[]	0
11	not_a_lab/2dash13dash2.cpp	#include <iostream>...	67e4eb9f8050fdb1c6d175a0633f9151	[219,1]	[28]	28
12	not_a_lab/...	#include <iostream>...	67e4eb9f8050fdb1c6d175a0633f9151	[219,null]	[]	0
13	not_a_lab/while.cpp	#include <iostream>...	0429c4295842ee32a971979e2982431f	[139,1]	[10]	10
14	not_a_lab/getEachDigit.cpp	#include <iostream>...	0429c4295842ee32a971979e2982431f	[139,null]	[]	0
15	not_a_lab/printPowers.cpp	#include <iostream>...	5609a9b6fad6758db8bdc7bfe8abdad6	[182,1]	[8]	14
16	assignment1/main_1.cpp	#include <iostream>...	681b3c6c8d733dd67e2cf8cbf18e72ad	[185,1]	[5,6,12,13,14,15,26,27,28,29,30,31,32]	32
17	assignment1/main_2.cpp	#include <iostream>...	8c4b61763154ee7dc3acf2d81882a424	[185,1]	[1,2,4,5,6]	6
18	assignment1/main_3.cpp	#include <iostream>...	886eb972536975c81d97c533840c4941	[184,1]	[3,4,5,6]	6
19	lab-01/projects/hello.cpp	#include <iostream>...	dceeb5e909ded7fae7dc72a6115349a2	[89,null]	[]	0
20	assignment1/main_4.cpp	#include <iostream>...	aef1a15473e35e820282cc7d60b14828	[590,1]	[28,29,30,32,33,34]	34
21	assignment1/main_5.cpp	#include <iostream>...	4b28825c35faa03ae42034f051ed892e	[213,1]	[1,5,12,29,30,32,33]	33
22	Untitled1	#include <iostream>...	e8311408fbb994b6e9da9134cd5cd922	[1,1,18,null,1,1,148,null]	[1,2]	2
23	unt.cpp	#include <iostream>...	2a58d218e2042d31910da13fde34cc60	[190,1]	[1,3,4]	4
24	assignment1/main_6.cpp	#include <iostream>...	87f0475120c3e21950b0da2a129a7531	[312,1]	[2,10,11]	11
25	assignment1/main_7.cpp	#include <iostream>...	626a74b4b2524df57063fca83566ae60	[269,1]	[2,12,13]	13
26	assignment1/main_8.cpp	#include <iostream>...	b43dfe442dafbe986b89a15fb313518a	[221,1]	[14,15,16]	16
27	not_a_lab/...	#include <iostream>...	b26990c261add1c81a222838d60ad368	[163,null]	[]	0
28	assignment1/main_9.cpp	#include <iostream>...	f105acedca488c968dac0039683aa5dd	[381,1]	[16]	16
29	assignment1/main_10.cpp	#include <iostream>...	8a20b013363002bbf5b40d9887b2cd94	[897,1]	[10,11,12,14,37,38]	38

Figure 3. This is a screenshot of the database file provided by the CS50IDE software.

3.2.3 Testing the Scale

While accessing the history of one account was straightforward, a larger trial group was needed to continuously test the practicality of implementing "sharing your work-space" in an entire classroom. I gathered a group of five computer science students and instructed them to share their accounts with my research account. The main account successfully received every request, emailing the corresponding email address with information on how to connect to the new student environment. Opening every account to test access was proven successful for each student. Each environment shared student files as well as the back-end CS50IDE hidden files. None of the test group members participated in the course pilot.

3.2.4 Deploying

After completing these tests, the project's focus transitioned to the target audience. CSC 211, a beginner object-oriented programming course, was the course used to implement this logging. This group was chosen because they use the desired IDE and they are an introductory 200 level course. By selecting a less experienced class, we hoped to find a good range of techniques in student programming. This class received a form to provide consent and voluntarily sign-up to participate in the research. Students were not provided any incentive to participate in the project and their data was to be made anonymous after attaching their data to assignment grades. Course instructors and TA's could also not have access to the data until after course grades were finalized. CSC 212, a data structures and abstractions course, was also invited to participate in the study. Students were invited to participate, although the course did not support the use of CS50IDE. 29 students signed up to participate in the pilot study.

3.2.5 Analyzing the Data

The data from the student assignments was analyzed using scripts I have created in Python. Using the built-in SQLite package in Python, the raw database was converted into readable data. I converted the database into a simpler dictionary data structure for easier manipulation in Python. This was also done to prevent changes to the original file provided. Personal identifiable information was also taken out at this time. This included things like private hidden files, other shared accounts, and chat messages.

3.2.6 Assignment Downloads

After each assignment was due, I performed another download of the students' history file. This would allow for more specific analysis so that calculations could

be made to determine work done over a project's assigned time. These files were named with the students GitHub names for identification. This information was removed once the grades have been received for that specific assignment. These downloads occurred three days after an assignment was due to capture possible late submissions.

3.3 Building PAPI

3.3.1 Study Design

I conducted interviews with 15 students/instructors with experience in programming and knowledge of computer science concepts.

Interview Style

Semi-structured interviews were held so that a conversation style survey could be had with participants. Removing the formal style related to other surveying methods allowed for a *discussion* of PAPI's implementation so that the user could express their ideas freely. Questions could be answered in a way best understood by the participant and opinions could be accurately provided.

Interview Members

To create the PAPI software, I first had one-on-one interviews with members of the University of Rhode Island community. This included current, past, and future instructors. This group were instructors; consisting of current, future, and past lectures and Teacher's Assistant. This community is the potential target user of PAPI when deployed. Each survey took place in about twenty minutes, with the participants answering questions related to who they are and what they would assume PAPI would be capable of. The group was not told specifics until additional ideas could be brought forward of what type of student data they might be interested in receiving. PAPI's frontend and operations were then built to

match their specifications.

Participant Recruitment

Students and teaching assistants were recruited via university channels. From this original group, more individuals were added by snowball sampling. To be eligible to partake, participants must have been at least 19 years old. They were also required to have some programming experience in any programming language. Teacher's Assistant (TA) experience was preferred but was not required. There was no compensation for participating. To sign up for the interview the individual would email the student investigator to schedule a time free in their weekly schedule.

Interview Design

Interviews were held using the video conferencing software, Zoom. This software allows for participants to make traditional or internet calls to the interviewer. To keep all interviews in the same format, the video portion of the software was disabled for the duration of the interview. These interviews originally were to take place in-person but were changed to an online setting due to "social distancing" laws. The interview was divided into four parts and took about 30 minutes (21-52 minutes). The questions can be found in Appendix C. The first part was intended to obtain a background of the user's experience with grading software. Questions were asked about auto-grading software because this is the closest related idea to what we are trying to implement. It is assumed that many people have experience with auto-grading software and not have had auto-feedback software experience. We wanted to understand the user's mental model of PAPI so that we can build to these expectations. We also wanted to know what they did not like about the current systems and how ours can overcome these issues. This was followed by

a programming session, where the user would code a small program. The task included coding a simple function checking if the input is an even number. This helped the participant get in a programming state of mind before reflecting on what information they deem important for the PAPI software. The programming challenge was given to see if there are any metadata that we should be collecting that we may have not thought of prior. In this same section, we had the participants rate our ideas for how to measure student work. The third stage of the interview was about designing PAPI. This was done in terms of how visual elements should be deployed. To understand the participants' background, the survey concluded with demographic questions. While demographic questions can be done at any point of the interview, a study found this made minority groups under perform [1]. The participant was allowed to ask questions throughout the process. The interview was concluded after any participant follow up questions were addressed by the interviewer.

When interviewing, the participant questioning started vaguely and then became progressively more specific in each section. To start, the questions were to collect participants' ideas. The goal was to utilize the very minimal instruction so the participants may generate ideas that we have not thought of previously. This allowed us to broaden the scope of the PAPI software for a diverse audience. When brainstorming I am aware of the limits in the CS50 software, possibly leading to the absence of user-desired features. Due to the participants' limited understanding of CO50IDE and PAPI, they were able to think of more abstract thoughts. The interview process continued by educating the participants on most concepts of PAPI, and things we wanted to implement. This step allowed for them to think of other related notions and rate our opinions.

Participant Demographics

The survey totaled 15 individuals (8 male and 7 female). Their ages ranged from 20 to 58 years old, with an average age of 25.4. The majors were 60% Computer Science, 33% Engineering, and 3% Writing and Rhetoric. The majority of participants (6/15) were TA's with an average of four years of experience. There were also four research assistants, three students, and two instructors.

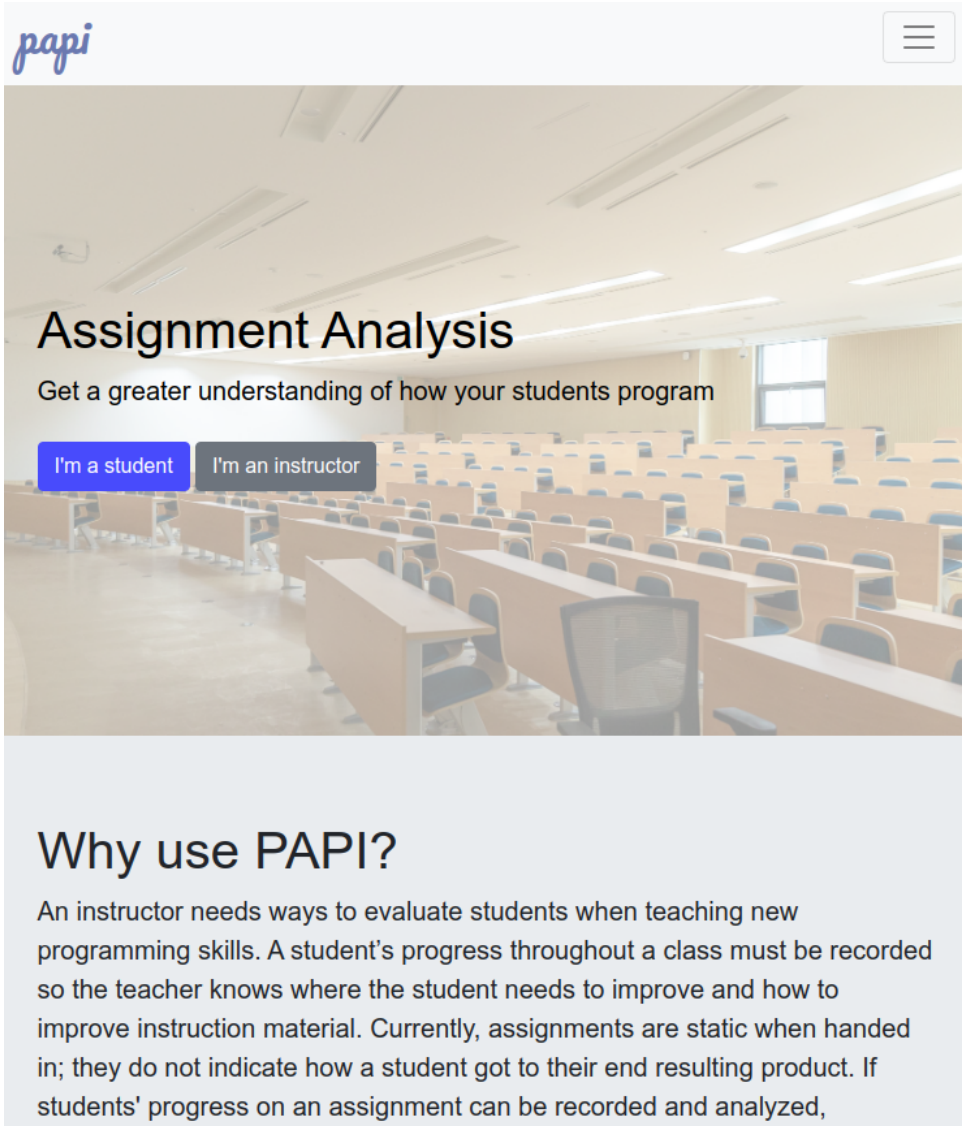


Figure 4. This is a screenshot of PAPI's homepage. From this page the user can chose to enter either student or instructor mode.

3.3.2 Creating the Software

After receiving the information the user base was interested in, we began writing the PAPI software. The software can be found online at https://github.com/DanielGauthier8/PAPI and was written by Benjamin Dahrooge and I. The PAPI back-end was written using Python3. We use HTML as a front-end and Flask as the web server gateway interface (WSGI). The project uses GitHub for version control and for managing collaboration. To start, the interface was written to allow for the upload of a single database file. This process continued to extract specific measurements and metadata to provide to the instructor. A mode where multiple files can be processed was added after single student analysis was implemented. More screenshots of the software can be found in Appendices A and B.

3.3.3 Data Parsing

The students' assignment database files were then uploaded into PAPI on a per student per assignment basis. I recorded the data from the PAPI software and changed student usernames to grades received on the assignment. Names were stripped from the recorded data after grade information was added according to the participant consent form. To remove cumulative numbers from each database collection, the data were corrected by subtracting prior data-points from each cumulative recording. This allowed each data point to be associated with a single assignment. After removing empty student submissions, 56 assignments were left and were used going forward. Comparisons were made to compare differences in grades and programming patterns. Averaging of student data was done in grade letter ranges. Thorough analysis lead to the generation of results to follow.

List of References

[1] J. L. Hughes, "Rethinking and updating demographic questions: Guidance to improve descriptions of research samples," *Psi Chi journal of psychological research.*, vol. 21, no. 3, pp. 138–151, 2016.

CHAPTER 4

Findings

4.1 Goals Revisited

4.1.1 Formative Feedback

Student File Analysis

Formative feedback is generated based on the data found in the individualized student databases. To provide feedback to the teacher there are two techniques: student mode and class mode. Student mode provides the highest level of detail, showing all forms of data that can be collected. This would include total time spent, start date, end date, number of character insertions/deletions, and number of comments on the assignments selected. There is a pulse graph showing when a student types specific input. If the user comments their code, adds logic, creates user output, or performs a calculation, this shows an increase in activity in the respective areas. The user can select to choose all student files or individually select specific assignments. A date-time range selection can also be used for more specific searches. Class mode shows to the user a similar graph as seen in the student mode. All students are summarized and a class pulse is generated. The class mode can show to an instructor when the students in a class are performing their work and in what order. This page also generates a downloadable zip file with formative feedback for every database uploaded. Once unzipped, these files can be provided to the student. Each PDF can be uploaded with the rubric for any assignment. This file contains the same items found when in student mode. PAPI provides feedback for the teacher on the scale of each student and of the class as a whole. It also automatically generates personalized feedback that can be provided to the student to self reflect on their work.

To measure the accuracy of PAPI's file analysis, 22 test documents were made

Table 1. Test Source and Numbering

Test Name and Source	**Test Number**
Assignment 1 (CSC211)	1
Booking a Room (Kattis Question)	2
Secure Doors (Kattis Question)	3
Beat the Spread (Kattis Question)	4
Turtle Master (Kattis Question)	5
Button Bashing (Kattis Question)	6
Antiarithmetic (Kattis Question)	7
Brexit (Kattis Question)	8
Above Average (Kattis Question)	9
Cuckoo Hashing (Kattis Question)	10
Death and Taxes (Kattis Question)	11
Bits Equalizer (Kattis Question)	12
2048 (Kattis Question)	13
Conformity (Kattis Question)	14
Squawk (Kattis Question)	15
Canonical (Kattis Question)	16
Swap to Sort (Kattis Question)	17
Not Amused (Kattis Question)	18
Non-boring Seq. (Kattis Question)	19
Paintings (Kattis Question)	20
Verify This... (Kattis Question)	21
Scaling Recipes (Kattis Question)	22

Table 2. Date Testing

	Manually Recorded Notes		PAPI Results	
Test Number	**Start Date**	**End Date**	**Start Date**	**End Date**
1	6/6/20	6/15/20	6/6/20	6/15/20
2	6/4/20	6/4/20	6/4/20	6/4/20
3	6/4/20	6/4/20	6/4/20	6/4/20
4	6/3/20	6/3/20	6/3/20	6/3/20
5	6/3/20	6/3/20	6/3/20	6/3/20
6	6/18/20	6/18/20	6/18/20	6/18/20
7	6/22/20	6/22/20	6/22/20	6/22/20
8	6/22/20	6/22/20	6/22/20	6/22/20
9	6/23/20	6/23/20	6/23/20	6/23/20
10	6/24/20	6/24/20	6/24/20	6/24/20
11	6/24/20	6/24/20	6/24/20	6/24/20
12	6/25/20	6/25/20	6/25/20	6/25/20
13	6/29/20	6/29/20	6/29/20	6/29/20
14	7/1/20	7/1/20	7/1/20	7/1/20
15	7/1/20	7/1/20	7/1/20	7/1/20
16	7/1/20	7/1/20	7/1/20	7/1/20
17	7/1/20	7/1/20	7/1/20	7/1/20
18	7/2/20	7/2/20	7/2/20	7/2/20
19	7/2/20	7/2/20	7/2/20	7/2/20
20	7/2/20	7/2/20	7/2/20	7/2/20
21	7/3/20	7/3/20	7/3/20	7/3/20
22	7/3/20	7/3/20	7/3/20	7/3/20

using object oriented challenge problems. The tester was an undergraduate student external to the software development. This was done to get real word results and help remove self testing biases. The first challenge problem was assignment one from the University of Rhode Island's Object Orientation course. The others came from the Kattis Archive site[1]. Please reference Table 1 for the source and name of each test.

The data to be measured was manually recorded and then compared to the PAPI software. First a test of the date metric was checked for basic data retrieval. The software does not have to do any data parsing and only performs a simple lookup in the files database. All start and end dates were retrieved successfully for

Table 3. Manually Recorded Assignment Details

Test #	# of Comments	Time Worked	# of Sessions	# of Saves
1	125	4:35:01	4	72
2	5	0:05:51	1	1
3	11	0:02:40	1	3
4	7	0:07:04	1	3
5	4	0:00:10	1	1
6	2	0:02:06	1	3
7	0	0:00:06	1	3
8	0	0:00:19	1	3
9	0	0:00:17	1	3
10	0	0:00:42	1	3
11	2	0:00:04	1	3
12	0	0:00:16	1	3
13	29	0:36:37	1	11
14	6	0:13:20	1	4
15	12	0:22:39	1	6
16	8	0:14:10	1	6
17	7	0:09:06	1	5
18	10	0:12:44	1	3
19	4	0:24:25	1	8
20	4	0:18:55	3	6
21	6	0:09:38	1	5
22	3	0:10:04	1	5

each test (Table 2).

Next, data that has to be generated via other data were checked. The number of comments, time worked, number of sessions, and number of saves were manually recorded by a researcher in Table 3. This was done by journaling the number of sessions, the timing, and number of saves while programming. The results from PAPI matched most of these recordings with minor error; this data can be found in Table 4. The number of sessions and number of saves matched exactly to the manually recorded data. Total time worked was slightly lower for the average test on the PAPI software, at about 4.5% less than the time manually recorded. I assume this pattern is because someone working on an assignment will spend time at the beginning and end of each work session, not making any file edits. This

Table 4. PAPI Recorded Assignment Details

Test #	# of Comments	Time Worked	# of Sessions	# of Saves
1	125	4:33:02	4	72
2	5	0:05:41	1	1
3	11	0:02:40	1	3
4	7	0:06:53	1	3
5	4	0:01:00	1	1
6	1	0:01:00	1	3
7	0	0:01:00	1	3
8	0	0:01:00	1	3
9	0	0:01:00	1	3
10	1	0:01:00	1	3
11	0	0:01:00	1	3
12	0	0:01:00	1	3
13	29	0:32:30	1	11
14	6	0:11:47	1	4
15	14	0:22:13	1	6
16	8	0:13:27	1	6
17	7	0:07:49	1	5
18	10	0:09:14	1	3
19	4	0:22:30	1	8
20	4	0:15:47	3	6
21	6	0:06:32	1	5
22	4	0:08:18	1	5

time is then not counted in the PAPI software's timer. This would be counted by a manual timer and arguably still considered time worked. It can also be observed that the assignments with extremely small work times were automatically given one minute. When there are too little text insertions to provide an accurate reading, PAPI automatically assigns one minute of work. When calculating the number of comments there was also a slight error of 0.4%. This means the PAPI software on average indicates 0.4% more comments than that actually found in the file. This slight error is because of the many variations a comment may be written depending on the programming language. PAPI can improve this error rate by changing what comment structure to look for depending on the programming language used.

Plagiarism Indicator

Plagiarism indication has been built into the student mode as well as the class export function to catch large copy and pasting instances. The indicator provides the file name, finds the pasted text, and provides the text identified as pasted. PAPI starts searching for pasted text after the first text insertion until the most recent edit. It would be assumed this first text insertion is the assignment instructions or starter code. Files written outside the IDE and uploaded are also automatically considered an approved outside resource. PAPI will start looking for pasted code in all edits after this upload. If a student pastes in another student's work and submits it as their own, this would be caught by any other cheating identification software. Therefore the PAPI software ignores this form of plagiarism. To identify work as cheated, PAPI calculates the average text inserted per database entry. As the CS50IDE saves at a consistent rate, we can identify outliers. If a student has a piece of text entered that is over 8 times their traditional cadence, then they are flagged for possible cheating and the text is shown on PAPI's dashboard. PAPI also flags entries that have text insertions above 400 characters. This was added to identify a file with too few text insertions, or if all text is pasted at a high speed.

PAPI was tested with multiple example student files with assorted sizes of copy and paste lengths. This can be viewed in Table 5. In our tests, all pastes were detected from character length of 50 to 3260. PAPI was also run on assignments without any copy and pasting. It can be seen in Table 6 none of the assignments written out by hand had large text insertion detected. The first pasted comments from one test run were also properly ignored, as the PAPI software skips the first text insertion. False positives and false negatives were not observed in either of these tests. A larger test group will be needed to find edge cases. According to

Test #	Programming Method	Large Text Insertion
1	First comment pasted in, Written by hand	FALSE
4	Written by hand	FALSE
14	Written by hand	FALSE
15	Written by hand	FALSE
16	Written by hand	FALSE
17	Written by hand	FALSE
18	Written by hand	FALSE
19	Written by hand	FALSE
21	Written by hand	FALSE
22	Written by hand	FALSE

these tests PAPI has 100% accuracy in determining copy and pasted code. No false positives or fast negatives were recorded at this time in PAPI's tests. Going forward it should be determined how many of these pasted code cases are related to plagiarism to get a full picture.

4.1.2 No Workload Increase

The generation of student feedback is immediate for the majority of situations using PAPI. When in student mode, the generation is always under one second. This operation is typically faster than one second, at around 400 milliseconds. Please reference Table 7 for all time measurements. To represent the largest a database might get, these values were recorded with the 10 largest databases of the semester. The number of clicks to use both student and class mode can be done in under seven clicks. Student mode can be completed in six clicks total and class mode can be completed in five. PAPI's checks will take place in its all in one software without the need for looking over any specific code. This can act as a tool for new TA's who are unfamiliar with helping other students as well as a center of knowledge for those with more experience. To sit down with a student in office hours, a session with a single student can range from about five minutes to the entire session, at around three hours. Getting insight into the student code can be done in these sections to provide specific comments. To ask a student their total

Table 5. Copy Paste Detection Testing

Testing File	Paste Size (characters)	PAPI Large Insertion Detection
2048.cpp	320	CAUGHT
	50	CAUGHT
	200	CAUGHT
	3260	CAUGHT
conformity.cpp	320	CAUGHT
	350	CAUGHT
	500	CAUGHT
	3260	CAUGHT
squawk.cpp	320	CAUGHT
	650	CAUGHT
	800	CAUGHT
	3260	CAUGHT
canonical.cpp	320	CAUGHT
	950	CAUGHT
	1100	CAUGHT
	3260	CAUGHT
swaptosort. cpp	320	CAUGHT
	1250	CAUGHT
	1400	CAUGHT
	3260	CAUGHT
notamused.cpp	320	CAUGHT
	1550	CAUGHT
	1700	CAUGHT
	3260	CAUGHT
nonboring.cpp	320	CAUGHT
	1850	CAUGHT
	2000	CAUGHT
	3260	CAUGHT
paintings.cpp	320	CAUGHT
	650	CAUGHT
	2150	CAUGHT
	2300	CAUGHT
	3260	CAUGHT
queens.cpp	320	CAUGHT
	2450	CAUGHT
	2600	CAUGHT
	3260	CAUGHT
recipes.cpp	320	CAUGHT
	2750	CAUGHT
	2900	CAUGHT
	3260	CAUGHT

Table 6. Time Measurements of Processing Database Files

Database Size (MB)	**Test 1 (ms)**	**Test 2 (ms)**	**Test 3 (ms)**
2.1	506	494	508
1.8	276	280	274
1.7	384	389	347
1.2	437	419	417
1.2	454	410	428
1.1	350	354	338
1.1	464	443	460
1.1	322	322	339
0.9	337	355	379
0.8	269	218	225

time spent, start date, end date, number of deletions, and number of comments in an assignment would be estimated to take at least five minutes for a student and there would be a large loss in accuracy. The PAPI software provides a lot of added benefits without the added time that would be traditionally required.

4.1.3 Have a Wide Impact

The possible impact of PAPI is large on both the university scale and the national level. The number of potential students for impact was determined by looking at the number of students and classrooms that use the CS50IDE. To have a wide impact, PAPI was written to work with cloud9 CS50IDE. The CS50IDE software is free and only requires a GitHub account to sign-up. CS50IDE is currently being used in the University of Rhode Island's sophomore curriculum for the object oriented programming and data structures class. These classes traditionally have at least 100 students registered. By teaching the students this IDE earlier in their schooling careers, the students can continue using this software for future classes like Computer Organization, Operating Systems and Networks, and Design and Analysis of Algorithms. At a minimum, assuming students do not continue using the same IDE after the required two courses, PAPI can reach the goal of 500 students in only a year and a half. The CS50IDE has similar popularity

country-wide as it does at the University of Rhode Island. The CS50IDE software, according to their systems administrator, has 150,000+ users in the most up to date version of the software [2]. This tool can be provided to these users with no additional changes to their workflow. With this number of current users I will also be able to meet my goal with 10,000 nationally in about a month. The last version came out Jul 5, 2019, a year before this reading was taken, reducing to 15,000 new users a month[3].

4.2 Software Results

Results were first obtained from the Google Form filled out by the interviewer, as found in Appendix C. Upon completion of all interviews, the results were compiled in a Google Sheet in order to code each participant response. The coding process worked by looking through user responses from each question, taking out all the main ideas. Each response was not limited in the number of codes they could receive, having 1-5 codes per response. If many participants mentioned a subtopic of a code, the response would receive both the overarching and subtopic code. Any unrelated information provided by the user for each question was put into its own category to act as a list of notes to keep in mind while developing PAPI. For the purpose of this paper, the participants will be referred to as P1 through P15.

4.2.1 Previous Experience

To understand each participant's background with software grading, questions were asked on previous interactions with auto-graders and grade-book programs. The two groups that participated were students and instructors. 86.7% of the individuals interviewed had experience with software grading. Of all participants, 40 % had experience as a grader in these systems. This is desirable, as the target

Table 7. Opinions Towards Current Grading Software

Opinion	Number of Participants
Fast feedback	9
Fail tests for not a good reason	8
Poor display of feedback	6
I can check my answers while working	6
Easier for the TA	5
Provides better feedback than a person	4
Bad inter-compatibility	4

audience should have previous experience in general. In addition, having both grading and student experience helps diversify answers. Opinions related to auto-grading that four or more people mentioned can be viewed in Table 8.

Current opinions

As shown in Table 9, the most liked thing about auto-graders was the fast speed of receiving feedback. This was mentioned by 60% of the total participants. The next largest positive to software grading is that it can provide better feedback for the students than human graders can. Students mentioned that while the software grading feedback is traditionally vague, it is typically more itemized than some professors' grading schemes. Students are able to see exactly what was wrong, item by item. The TA's mentioned a similar improvement, saying that they can grade more assignments in less time when a large portion is auto-graded. This allows them to spend more time critiquing those who scored poorly.

Issues with software grading consist mostly of having assignments marked wrong for poor reasons. 53% of participants mentioned this problem in different ways. With auto-grading, there is a pattern for requiring very specific output of a program and cannot provide partial credit if this is not met. Sometimes the instructors write auto-graders incorrectly and the student is then notified that they did something wrong, even if what they did might have been correct. These

issues are highlighted further as vague feedback or no feedback is provided as to why an answer is incorrect. The key weakness is when an issue occurs and the student is not advised on how to fix it. While the points awarded are specific and considered a positive, when things do go wrong and a TA is not there to provide further information, the students simply get mad with the current implementation. It should also be considered that the student does not get overall feedback on how they worked on the assignment as a whole with the current system.

It is beneficial to understand where the PAPI software will fit into these current gaps in auto-graders. The main purpose of our software is to provide formative feedback and this is considered a largely lacking element by those who use auto-graders.

A common theme between these conversations was the relationship between the grader and the student. The approval and disapproval of software grading depended a lot on the participants' opinions on the importance of having this relationship. This was not a separate question but instead came up in many of the discussions. The dislike of software grading seemed to stem from the idea that when an instructor is grading an assignment a necessary relationship is built between the teacher and the student. It is through this relationship that the teacher learns where students are struggling and succeeding. Participants who liked software grading attributed to *not* having this relationship, allowing for a more even and fair grading system. It was mentioned by P9 that "I've had TA's make mistakes in the past", continuing to explain how TA's can add a bias to the grading without even realizing it. Software grading will grade every student with the same reasoning every time, giving each student a completely equal opportunity to succeed.

In regard to current software interfaces, a pattern in feedback came up again.

40% said that the feedback, when supplied, is displayed in a poor manner. The current systems make feedback, when provided, difficult to find when looking through grades. With current changes to the University of Rhode Island's LMS, it will be interesting if this opinion is kept going forward. The second complaint was poor inter-compatibility between grading software, coming up in 27% of the interviews. While the first iteration of PAPI is stand-alone, its future work involves integrating directly inside of an IDE. I kept this in mind as PAPI has an export function. While the front-end may change over time, the back-end is being written in Python for easy re-implementation in the future.

4.2.2 Programming Skills
Participant Ideas

When analyzing the students' key-logging information, I want to make sure I was capturing statistics that graders and students will find useful. I started by asking participants what students should do to ensure positive results on an assignment. This questioning was done to see if I could automate checks for each of these ideas. All responses are summarized in Table 10. Of these responses I will mention some that could be integrated into PAPI and how.

The idea of planning out an assignment before starting was brought up by 80% of the participants as an item I need to make a top priority. PAPI can implement a check like this by measuring when the student is writing out their pseudocode/comments. Comments written at the beginning of the process can be weighted more than at the end of the process. I also put a focus on how much pseudocode they are writing out, assuming that more pseudocode is better for retaining concepts. Additionally, it was important to analyze "incrementally building" and "starting early and often". These two measurements can be detected in similar ways. By noting the number of times a student works on an assignment

Table 8. Good Programmers

Advice	**Number of Participants**
Plans out program before starting	12
Incrementally build	8
Modular design	7
Reach out to TAs, teachers, and friends	6
Look over material again	5
Starts early and often	5
Put in effort	4
Does not cheat	4
Tests code	3
Searches online documentation	3
Fixes errors as they come up	2
Does practice programming	1
Does not skip class	1

and how much they add to the program during that time will dictate if they are dividing their work well. A student who starts as early as possible and has many separate sessions with equally divided work is considered a perfect score for this measurement. Another overarching and measurable concept is the idea that a 'good' programmer does not cheat. Cheating can be identified by looking for the copying and pasting of large amounts of code. This can be detected if the key-logger inserts a row with very large amounts of text. The logging software records input at a rate of close to once every three seconds, while the fastest English typist can type 216 words per minute. If more than 10 words are typed in one data collection the student would be flagged for cheating. This number will be adjusted as necessary with future research, but acts as an example of a rate of text input not achievable by a human using a keyboard. One interesting topic brought up by some participants was how students handle problems that they encounter when programming. This was an idea I had not previously considered. With ideas of growth mindset stressed at the University of Rhode Island, it is important to obtain an understanding of how students handle failure as well as

Table 9. Implementation Importance Rating

Metric	Average Rating
Time spent	4.67
Copied and pasted text	4.27
Used comments while coding	3.87
Order of code elements	3.87
Start Date	3.53

their response to improve. While this idea is something I want to put merit in, I am still hypothesizing the best way to capture it in software. One idea I thought of included measuring the number of saves in a small period of time. Little changes between saves can also be a method to identify possible struggles. Another idea, provided by P2, was to measure the number of deletions in a set period of time. This would show the student is taking action to fix a mistake.

Rating Ideas

After getting participants' ideas on ways a student work can be measured, I had them rate hypothetical programming practices. Of the concepts aforementioned, the participants most liked the idea of detecting cheating, having 60% of the participants view it as the most important identifier. While this is not the intention of PAPI, many saw it as its most desirable feature. P15 mentioned, "[it's] easy to defeat current cheating technology, even the good ones". Previously, this was not a highly prioritized item. After hearing this feedback, the feature was given more attention. As I have acknowledged, assignments can only be written in so many ways. Detecting copying and pasting can help differentiate false positives with other deployed systems. Red-flagging students suspected of cheating and then having a conversation can stop the bad habit before it becomes an issue. Participants rated plain metrics on the Likert scale to be collected and presented to the grader. Options that received an average rating of 3.5 or higher were the first to be implemented in PAPI. These ratings can be found in Table 11.

These data points make sense as elements that are useful on their own. Metrics such as time spent on an assignment and start date can indicate to a professor the workload they are assigning to their students. As mentioned previously, it was not predicted how high copy and pasting code would rate in the Likert scale question. The "number of saves" for the assignment was not deemed important, receiving an average of 2.2. This is contrary to seeing this data in the context of checking for other measurements like "what the student does when they encounter a difficulty." It was rationalized that when rating simple metrics, such as the number of saves, it would be difficult to understand abstract uses. If an item was proved to have worth in the "participant ideas" portion, the data point will still be implemented in PAPI. The number of saves will, therefore, be added to the list of implemented ideas. Items like the number of file creations, the number of grammar mistakes, and the naming of each file were assumed to score poorly. They do not provide a lot of useful information on their own or in more abstract ideas from the "participant ideas" portion. These items do not relate to the students' process in a way that we deem important for performing well on an assignment. A student can be a bad speller, name an assignment anything they want, and create many files, but this will not greatly impact the running of a program. At this time, they do not relate to participant ideas or our ideas and will not be developed for PAPI.

In general, taking these data measurements can be deemed as intrusive to the students. When asking what information should not be shared with an instructor, even if it helps the student, the opinion was this data collection is something that the student should consent to and not automatically implement across the class. For the current study taking place this is not an issue. The student already had to consent to participate in the study. When implementing in the classroom, this same style may be recommended and should be looked into further. Similar issues

Table 10. Application format

Deployment Option	Number of Participants
Website	11
Downloadable software	2
CLI	2
App Store	1

are occurring with turnitin.com[4]. For the time being, students have to share their files to allow a teacher to see the data. It is assumed the consent has already been provided. Professors should consider this as an opt-in and not a class requirement. Another topic mentioned by the participants (47%) was limiting recording to only the assignment and not allow the viewing of other windows. This will not be an issue as only text entered in the IDE is logged.

Feedback Action Item

One participant articulated our goals well, saying, "teach, do not monitor" (P14). This aligns closely with what PAPI should be used for, acting as a teaching aid and not as a 'tattler' system. If students do not perform well on an assignment *and* missed an element mentioned, the teacher can recommend the student to use a different method. If a student performs well on an assignment and misses an element, it will be assumed the instructor will have that student continue their current routine.

4.2.3 Software Design

Software design was subdivided into three sections, this was the application format, the presentation of files, and the presentation of file analysis.

Application Format

One quick but important section was the applications deployed format. Flexibility was something I saw important as one of our goals was to reach the most

number of people possible. The grading/TA position is also unique because it is not typically a long term job. While students graduate from college, the same professor can teach a class for many years. There is a lot of turnover of TA's and this needs to be kept in mind when creating the PAPI software. I want to make sure PAPI is in a familiar format and can be quickly taught to a new grader to fulfill this requirement. In reference to Table 12, 74% thought a website is the best technique to achieve this. While there are many options for deployment, these are key deciding factors that need to be considered. With our premonition of using this deployment mechanism and the majority of participants marking this as the top option, I will make a website. A website is not bound by an operating system and can be accessed anywhere that has an internet connection. By running a web service on the grader's computer, similar to Jupyter Notebooks, PAPI could also run locally. By not requiring specific software, the setup process can be removed for the average computer user.

Presentation of File(s)

The interview became abstract when talking about how to display student files to the instructor. The database has a document ID for each file and records this with each user action. To be able to look through this database in a visual style is similar to looking through someone else's computer folders. This interface was created from scratch, so any recommendations would help with implementation. Although mandating specific formatting of the IDE folders is one way to help with this, I still wanted to see if there were additional implementations. The actuality will have to be a hybrid of having a good file viewer as well as requiring some structure for naming files. In terms of file sorting, the participants were told to imagine they were using another person's computer to find a specific file. They were then asked which steps they would use to find it and why. This was seen

Table 11. File Analysis

Data Source	Duration	Average Rating
One Student	One Assignment	4.53
One Student	All Assignments	4.53
All Students	All Assignments	4.53
All Students	One Assignment	4.27
Group of Students	One Assignment	3.47

as a parallel to traversing the students' database without having to relate directly to our specific software. The most popular option was to sort by last edit date, followed by having a visual directory tree. Having such a specific consensus, the goal of PAPI is to show files in a directory tree with the option to limit by creation date. This will allow for a swift process when looking at most recent assignments.

File Analysis

Table 13 shows which data source was most desired and at what granularity. The data source is the number of students analyzed at one time. The duration is the number of selected assignments, and the average rating is the popularity of participants who voted for the corresponding data source and duration. The participants were given options across axes; all students to one student and all assignments to one assignment. Choices were distributed in groups at either side of the matrix. To start, participants thought both the class as a whole and individual students should be allowed to be selected. This means there was no preference for deploying only class data or only student data. When analyzing the class as a whole, the participants deemed looking over every assignment as more important than one assignment at a time. When selecting one student, they had the opinion of both all assignments or one. The options for choosing a small group of students were not rated as highly.

In terms of implementation, these results are the best case. Having to select each student and then their specific assignment would have been challenging to

implement in an easy-to-understand way. The grader would have to select each student's assignment one at a time. By analyzing the entire class as a whole, the teacher can look at the overall trendline of the class for all assignments. This means they could see when students start working on new assignments and when they are putting in the most effort. This can help emphasize to students to start sooner or have the teacher change assignment deadlines. This same pattern would be useful for one student over a semester. If a student performed poorly on all assignments, there might be a clear pattern between projects. If a student performed poorly on one assignment, they might have used an unproductive technique. With this new data, I know PAPI should be able to support specific student patterns and general class patterns.

4.2.4 Implementation

The next step is to implement the ideas I learned from the target audience. With PAPI, there will be a limited amount of time before the end product can be created. These discoveries will help create a tiered system of implementation. Instead of assuming what action items should be deployed in the first launch, I now know what people want to be implemented and why. The research will also continue the process of analyzing student work. This included documenting previous work in computer science, as well as making discoveries of our own in the current parallel student study. Our process will have to repeat for the lifetime of PAPI if I want to continue to have a pleased user-base. Keeping up to date with current opinions and patterns is clearly a requirement for good software.

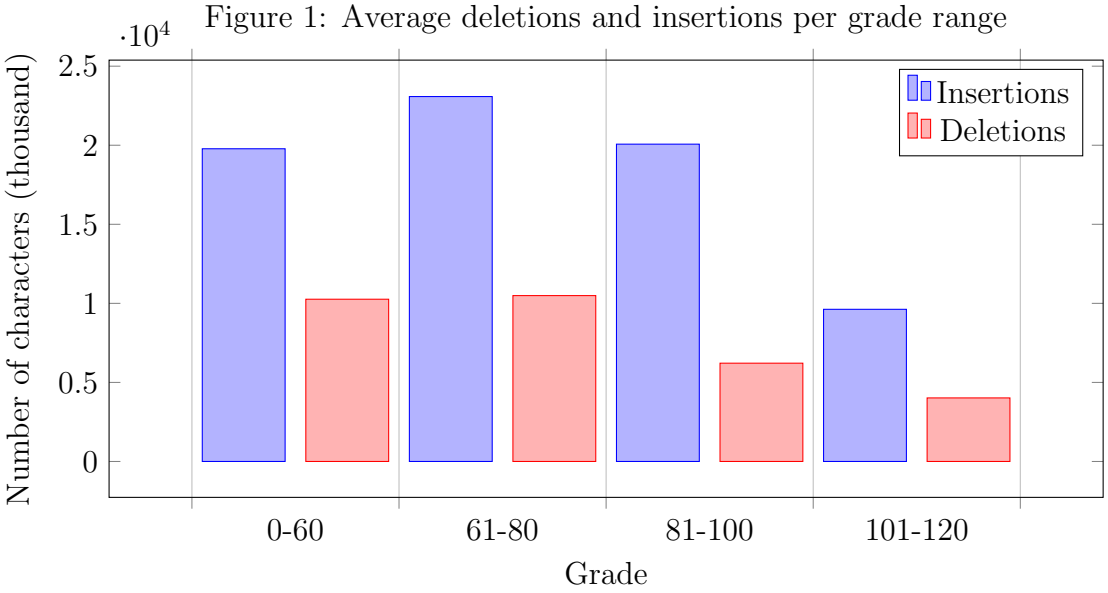
4.3 Student Patterns

I first looked at patterns comparing grades received on an assignment with a number of different metrics to measure results comparing grade and programming

techniques.

4.3.1 Character Input

Figure 1: Average deletions and insertions per grade range



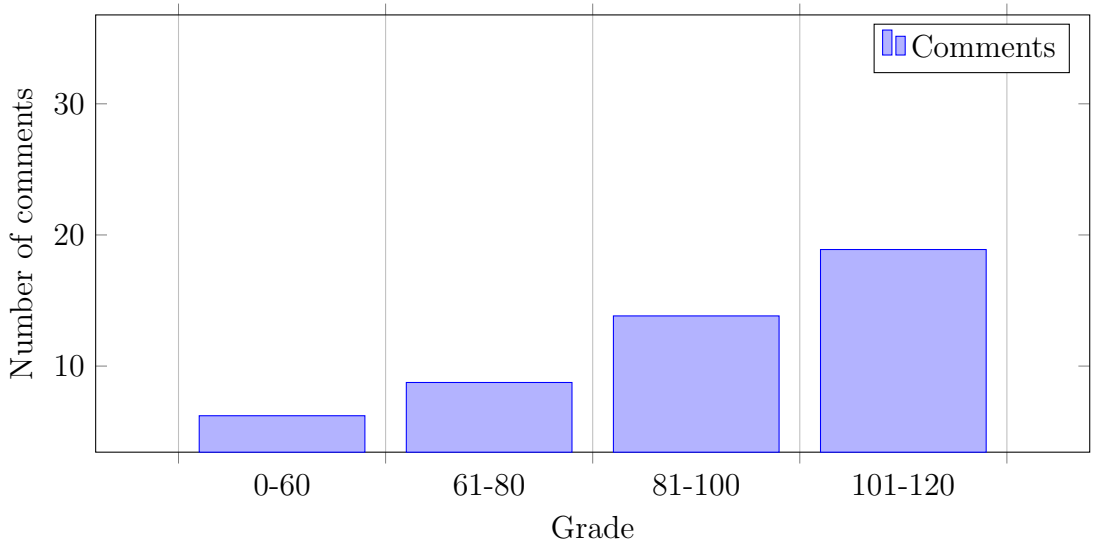
The number of insertions and deletions the student made over the course of an assignment submission period was recorded by PAPI. Numbers were generated based on the average at grade ranges of 0-60, 61-80, 81-100, and 101-120. All ranges were given equal bin size of 20, besides the first bin, as any grade below 60 is considered failing.

Figure 1 shows an average increase of 48.63 deletions and an average increase of 56.01 insertions with each jump in grade range. This means that the pattern of insertions and deletions have an inverse correlation with grade. Generally, when the number of insertions and deletions increases, the student's grade decreases. It can be noted this pattern is not found in those students who have received a B on the assignment. I assume this occurred because of the low sample size. There were only two students who received a B, making the results inconclusive for this grade range. A pattern was not identified when looking at the difference between deletions and insertions.

Table 12. Grade received vs last file creation and last edit

Grade Range	Average Last File Creation Time	Average Last File Edit Time
101-120	128:29:54	85:35:12
81-100	84:42:35	80:23:01
61-80	73:28:13	71:28:06
0-60	35:10:26	24:20:03

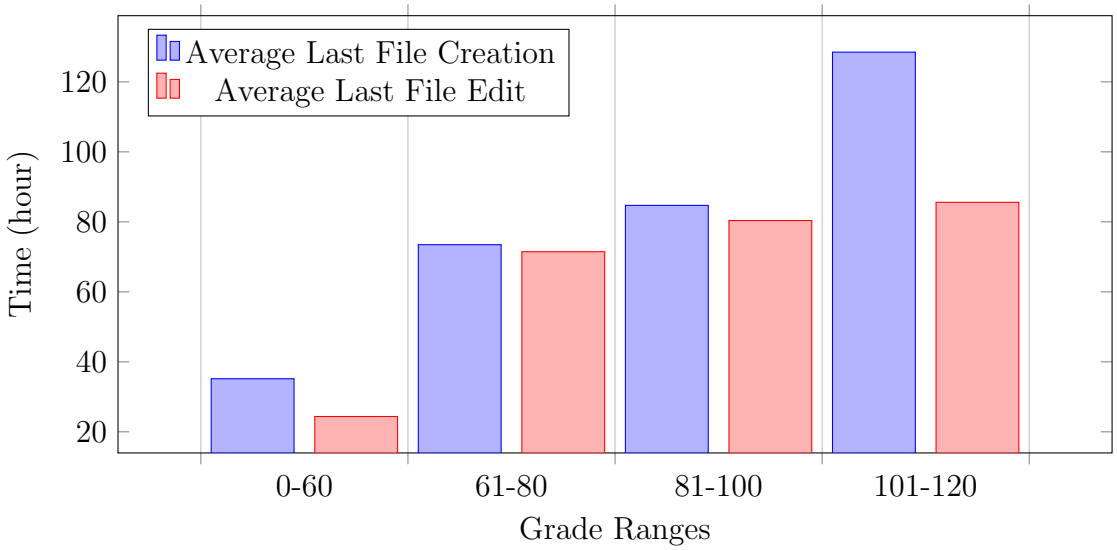
Figure 2: Average comments per grade range



Comments and grades were examined to find the impact of commenting code on the project grade. It should be noted that commenting was not an individually graded item on assignments. On average, it was found that students who received an A on an assignment commented more than any other letter grade. When comparing students who passed and failed an assignment ($\geq$ 65 considered passing), there were, on average, more than double the number of comments made by students who passed than those who failed (Passing average: 13.55 comments, Failing average: 6.21 comments). This shows that more comments typically lead to a higher assignment grade.

4.3.2 Timing

Figure 3: Grade received vs last file creation and last edit

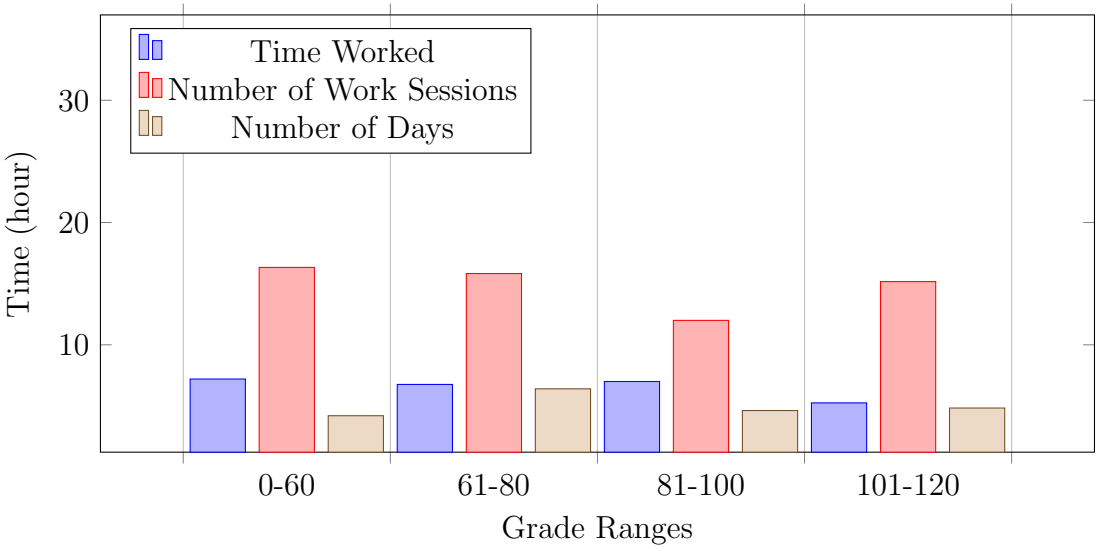


The time at which a student finished working on an assignment, in relation to its deadline, influenced the grade the student received on the assignment. The mean of students who received a grade of over 100 created all necessary files for a project early in their timeline, while those who scored lower created files closer to the submission deadline. This trend continues for the last edit time. Working on an assignment closer to the deadline led to a lower grade. On average, the last edit date for the students who received over 81 points had made their last edit earlier than lower-scoring students. The lower-scoring students had not created all of the files they needed for the assignment at this same time. One interesting find was that students who scored over 100 tend to have all files for an assignment made at least 9 days before the assignment was due. This does not mean that the assignment was completed this early, as the edit date continues to about 3.5 days before the deadline. This matches our assumption that students who start an assignment early and plan out their layout most often receive a higher grade than those who do not.

Table 13. Grade received vs time division

Grade Range	Time Worked	# of Work Sessions	# of Days	# of Saves
101-120	5:25:08	15.17	4.2	140.6
81-100	7:00:32	12.00	6.4	257.68
61-80	6:47:46	15.83	4.625	236.5
0-60	7:12:21	16.33	4.83	214.22

Figure 3: Grade received vs last file creation and last edit



The way a student divides their time on an assignment is another topic of interest. The time spent on an assignment did not vary drastically besides that of students who scored over 101. On average the students worked on the assignment for at least an hour and a half less than any other grade range, while still getting extra credit points. This leads us to believe that students who score very high on assignments either use the IDE not recommended for the class or they already understand the material so they are able to do assignments and their extra credit in a very fast time. The pattern at 100 and lower does not have a statistically significant pattern. This observation stands true to the number of work sessions and the number of days worked. Overall, there is little to no pattern found between grade and time spent working on assignments.

List of References

[1] "Welcome to the kattis problem archive." [Online]. Available: https://open.kattis.com/

[2] D. J. Malan, Jun 2020.

[3] D. J. Malan, "Cs50ide," Jul 2019. [Online]. Available: https://github.com/cs50/ide

[4] A. Davis. The Hoya. "Plagiarism-detection web site must not profit from students' work." October 2006. [Online]. Available: https://thehoya.com/plagiarism-detection-web-site-must-not-profit-from-students-work/

CHAPTER 5

Conclusion

This project has developed the PAPI software to provide data on how students approach programming and to support analysis of that data to help with formative feedback to both students and instructors. PAPI achieves this by providing measurement of time spent on an assignment, a copy and paste indicator, a comment counter, and a graphic of what students are working on during each programming session. The results from the course pilot found patterns showing the data that we collected has probable correlation between grades revived on an assignment and several assignment details. The PAPI software indicated in the pilot classroom that a low number of text insertions and deletions, starting an assignment early, and commenting code pattern to higher assignment grade. The data also showed students who receive grades over 100 have much lower usage of the course assigned IDE. These students had noticeably lower work time, number of edits, number of days, and number of saves. With both a desire from the community of this product and promising patterns in data, the PAPI software has a clear opportunity for deployment.

CHAPTER 6

Future Work

6.1 Software

Like all software, PAPI will need continuous updates to stay operational and relevant to current standards. This would be required to remain to its current security standards and progress with its dependencies. In addition to this upkeep there are more elements that could be added to PAPI. When writing PAPI, elements were introduced in a priority matching what our interview indicated. Some of these lower priority items on this list are not yet implemented. One example is to include the option of entering student mode from class mode. Another addition could be adding instructor settings that change how the website operates on a per-user basis. In addition, the PAPI software could save class patterns for comparing with students in future courses. While PAPI is written as an individual program to run alongside an IDE, the long term goal is to have this built directly into an IDE. An environment would be expected to handle the generation of the database, the retrieval of the history for the instructor, and the parsing of the data. This would remove the requirements of having to load the history file into a separate software.

6.2 Research

PAPI is a proof of concept to show different programming techniques and compare those with assignment grades. More time and attention is needed to perform an education and psychology study on which programming methods lead to higher course grades. Our proof of concept needs to be taken to a larger participant group. This group would be suggested to test large classes at different levels to see the difference in programming trends throughout the learning process.

APPENDIX A

PAPI's Software Interface

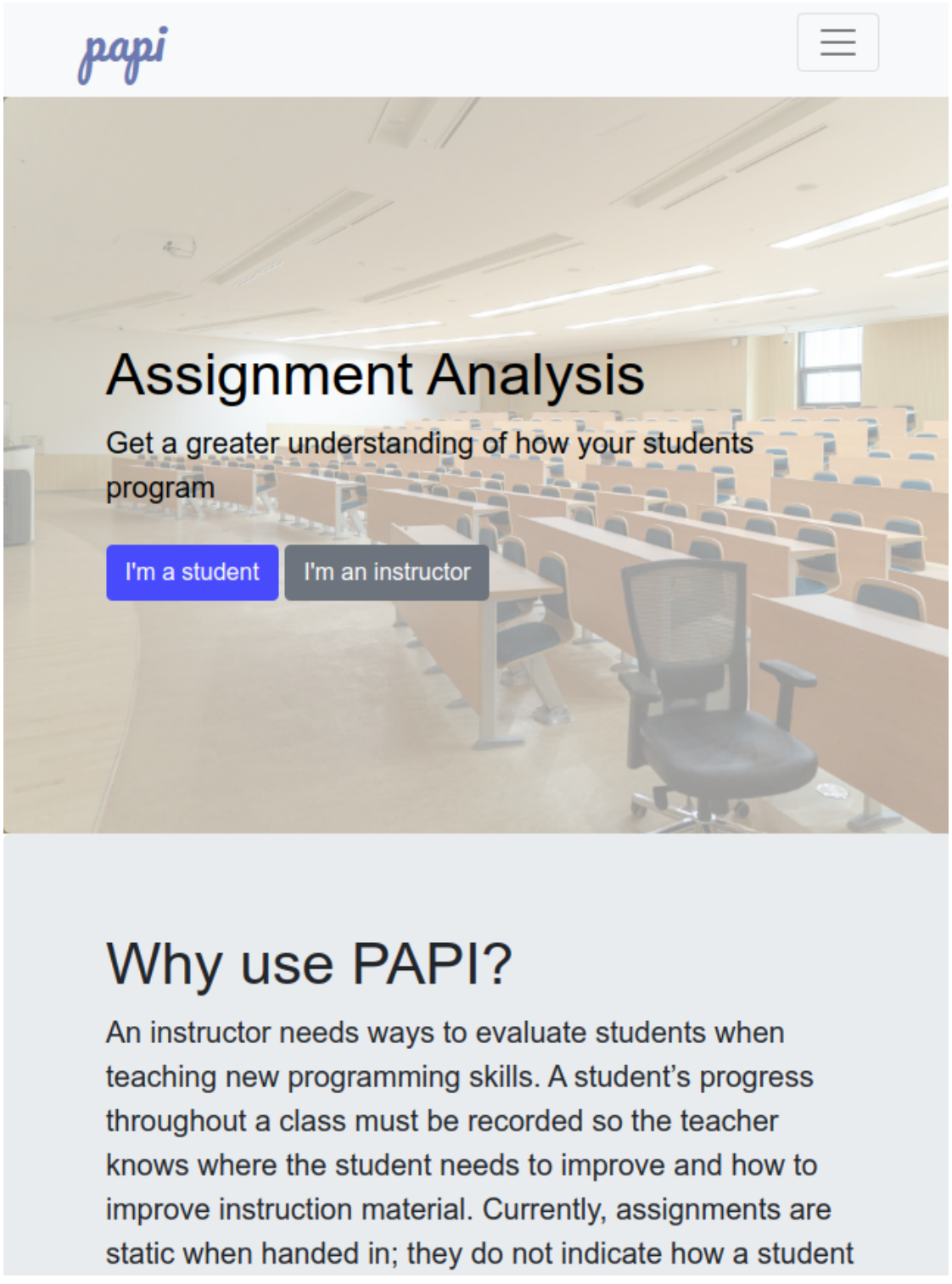


Figure A.1. The PAPI Home Page has options to use the single student mode or multi-student instructor mode.

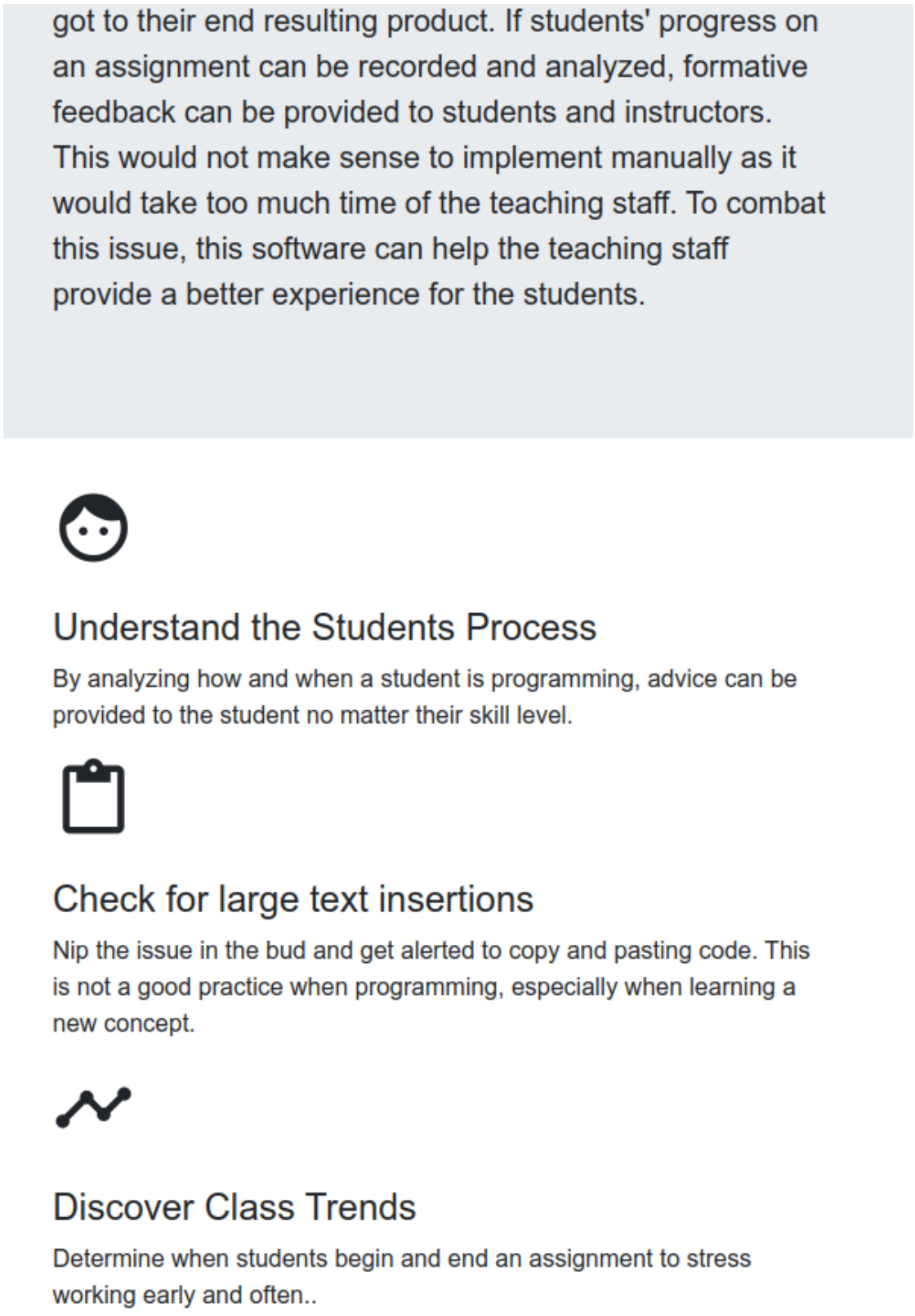


Figure A.2. The PAPI Home Page provides a brief overview of what the software can do.

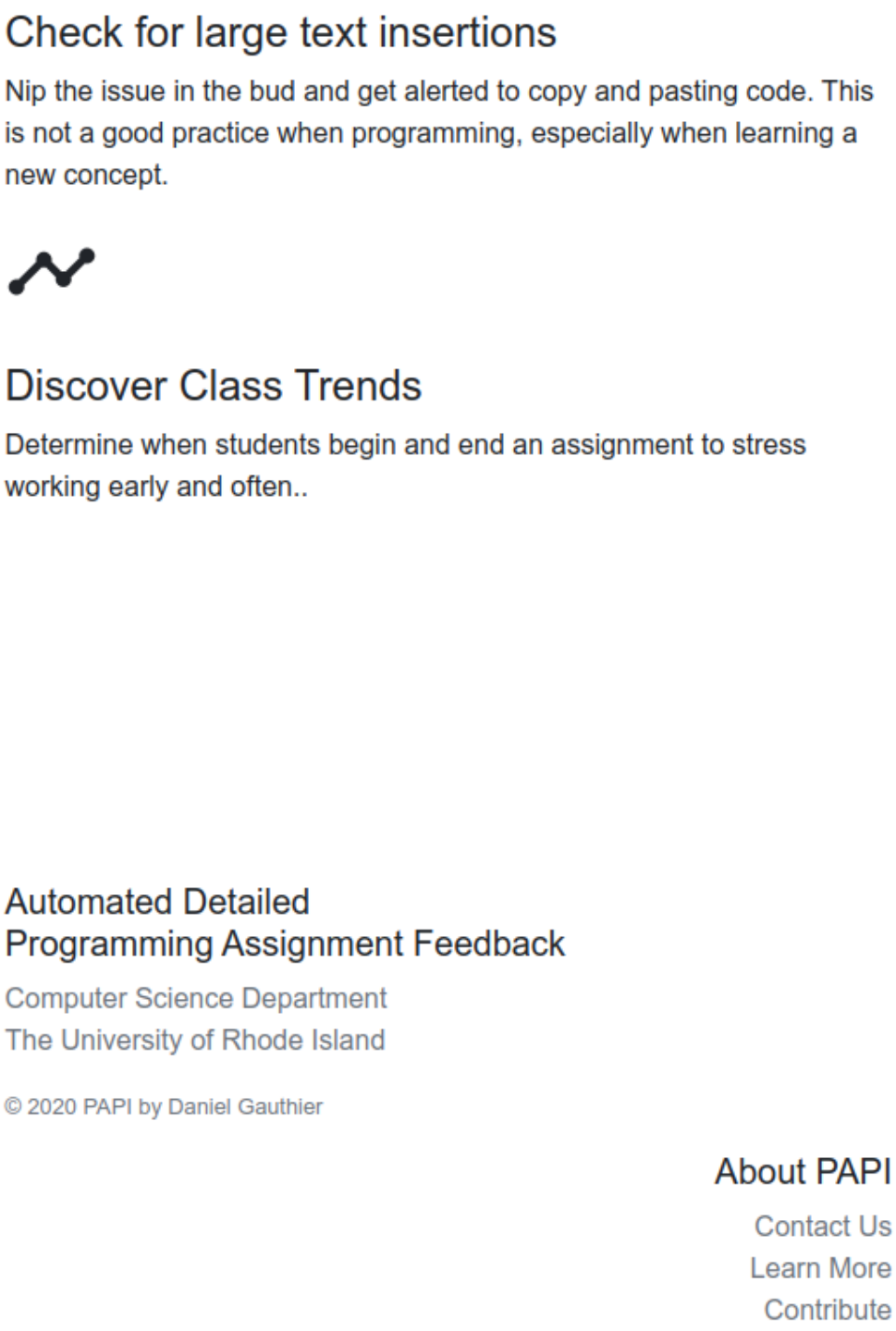


Figure A.3. The PAPI Home Page provides examples of what it can do.

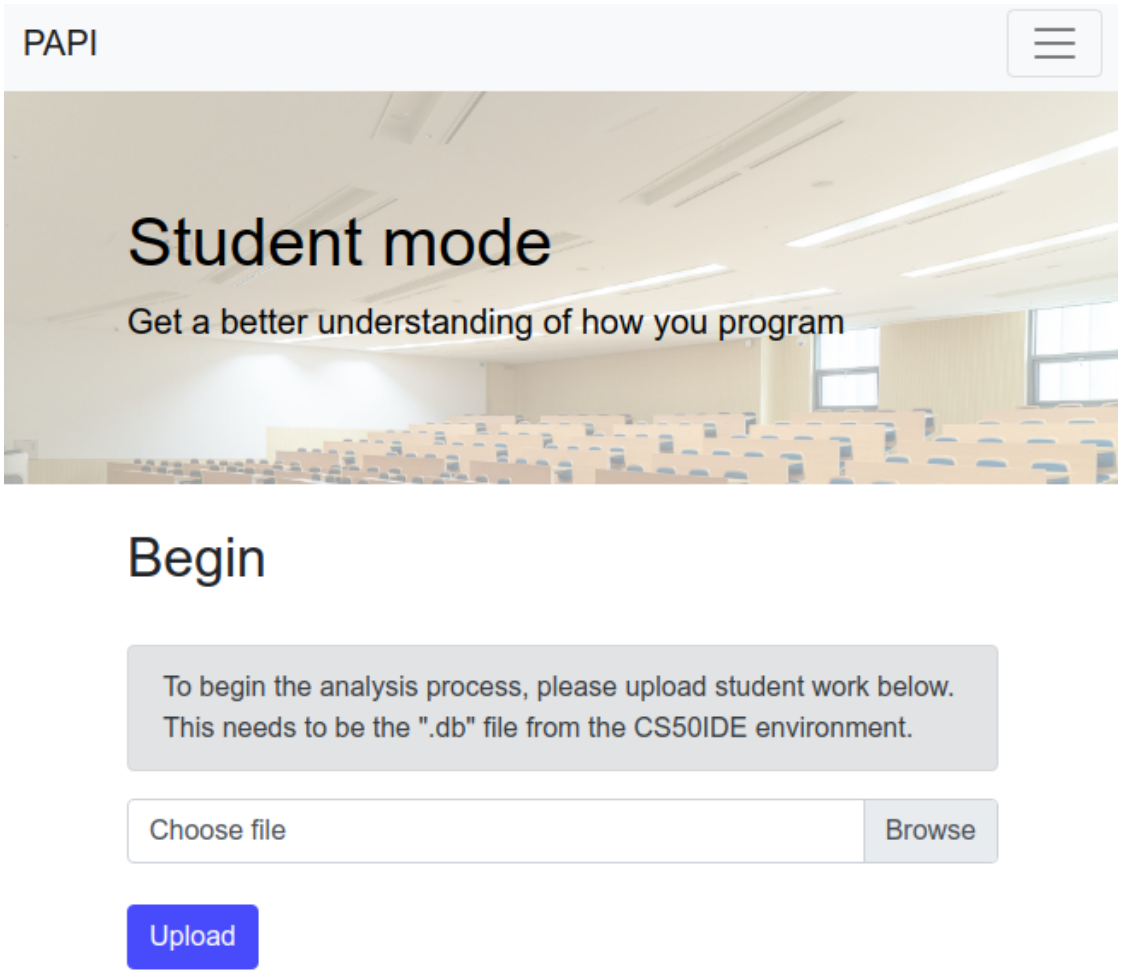


Figure A.4. The student mode file selection page allows for the upload of a single database file. By clicking on the browse icon the user can upload a database file from the CS50IDE software.

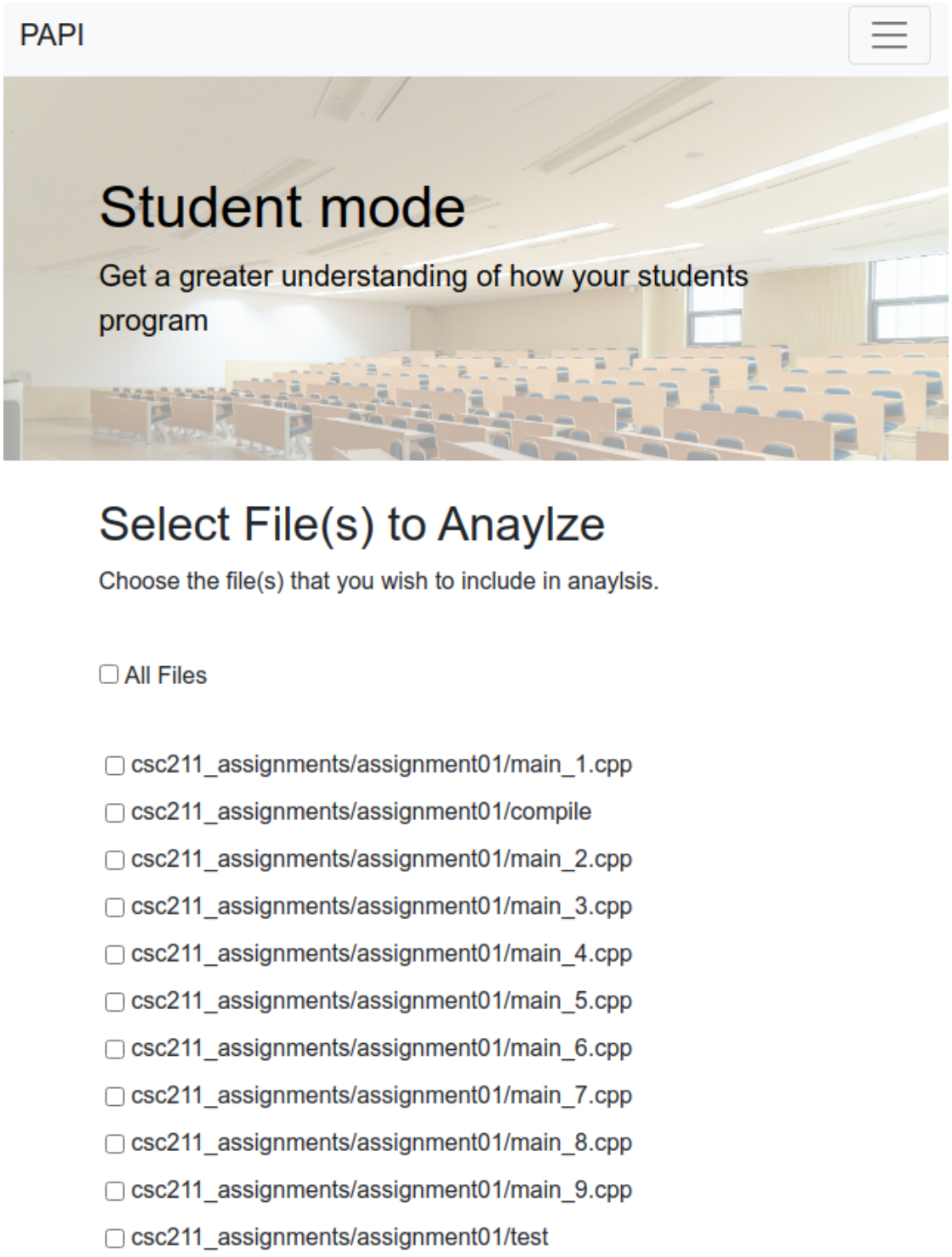


Figure A.5. The file selection page allows for selection by file or by date. Here is the file selection portion.

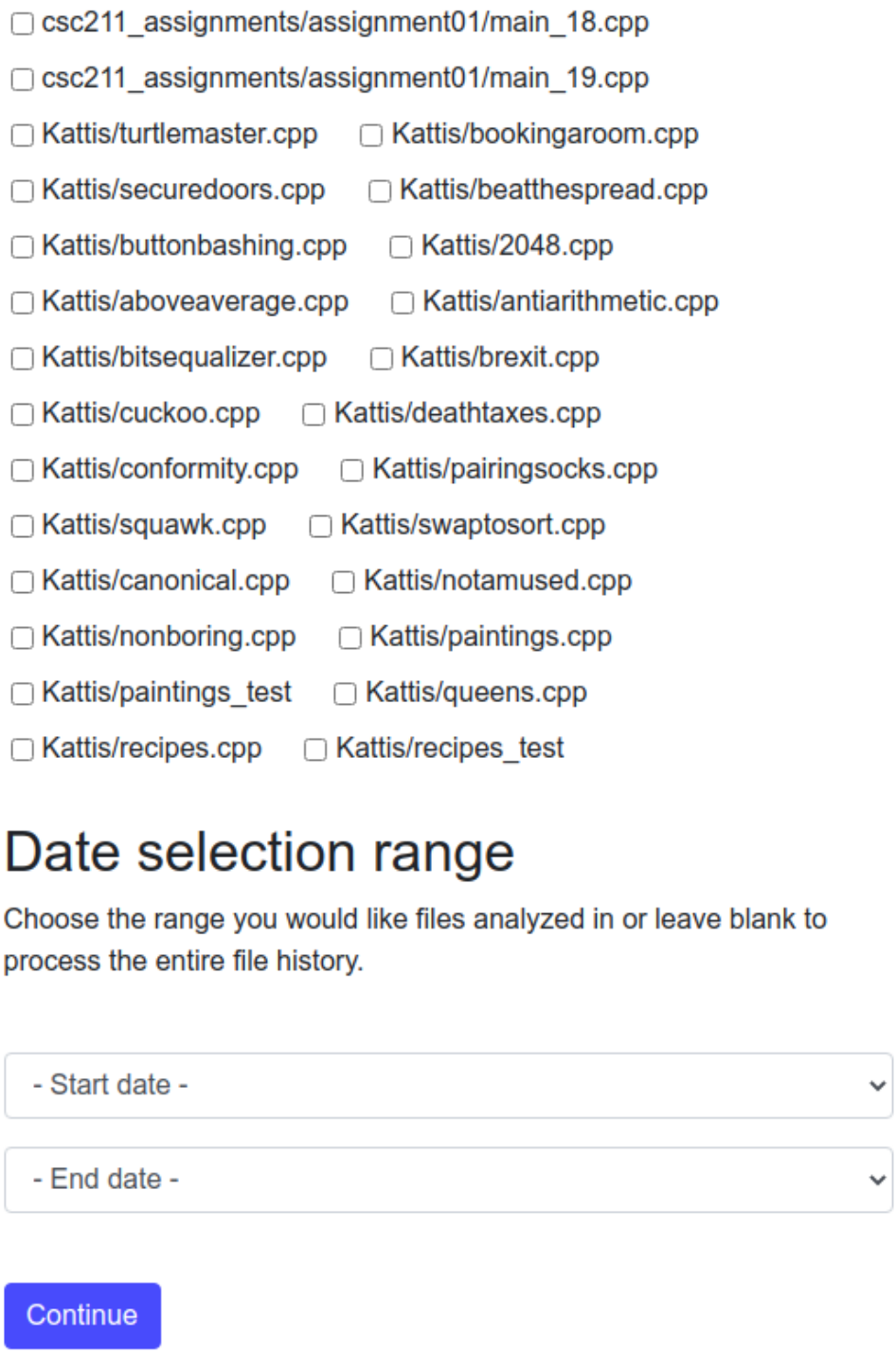


Figure A.6. The file selection page allows for selection by file or by date. Here is more of the file selection portion and the data selection section.

PAPI

File Data

File Name(s)	csc211_assignments/assignment01/main_1.cpp csc211_assignments/assignment01/compile csc211_assi *...(1369 characters)*
Time Worked	8:14:07
Number of Work Sessions	13
Over Number of Days	8
Average Work Session Length	0:38:00.538462
First File Creation Date	2020-06-06 21:27:58
Last File Creation Date	2020-07-03 16:37:13
Last Edit Date	2020-07-03 16:37:29

Figure A.7. The PAPI results page shows each metric with its matching value.

Last File Creation Date	2020-07-03 16:37:13
Last Edit Date	2020-07-03 16:37:29
Number of Saves	144
Number of Deletion Characters	2269
Number of Insertion Characters	34791
Number of Comments*	247

Large Text Insertion(s)

Kattis/buttonbashing.cpp	#include #include #include u *...(390 characters)*
Kattis/2048.cpp	for (int i = 0; i < 4; i++) { vector row; row.resize(4); *...(307 characters)*
Kattis/paintings.cpp	#include using namespace std;typedef long long ll;int n;int ways = 0;vector est *...(786 characters)*

Figure A.8. Here is more metrics as well as the large text insertion portion. The software shows the file the paste was don in and what text was pasted.

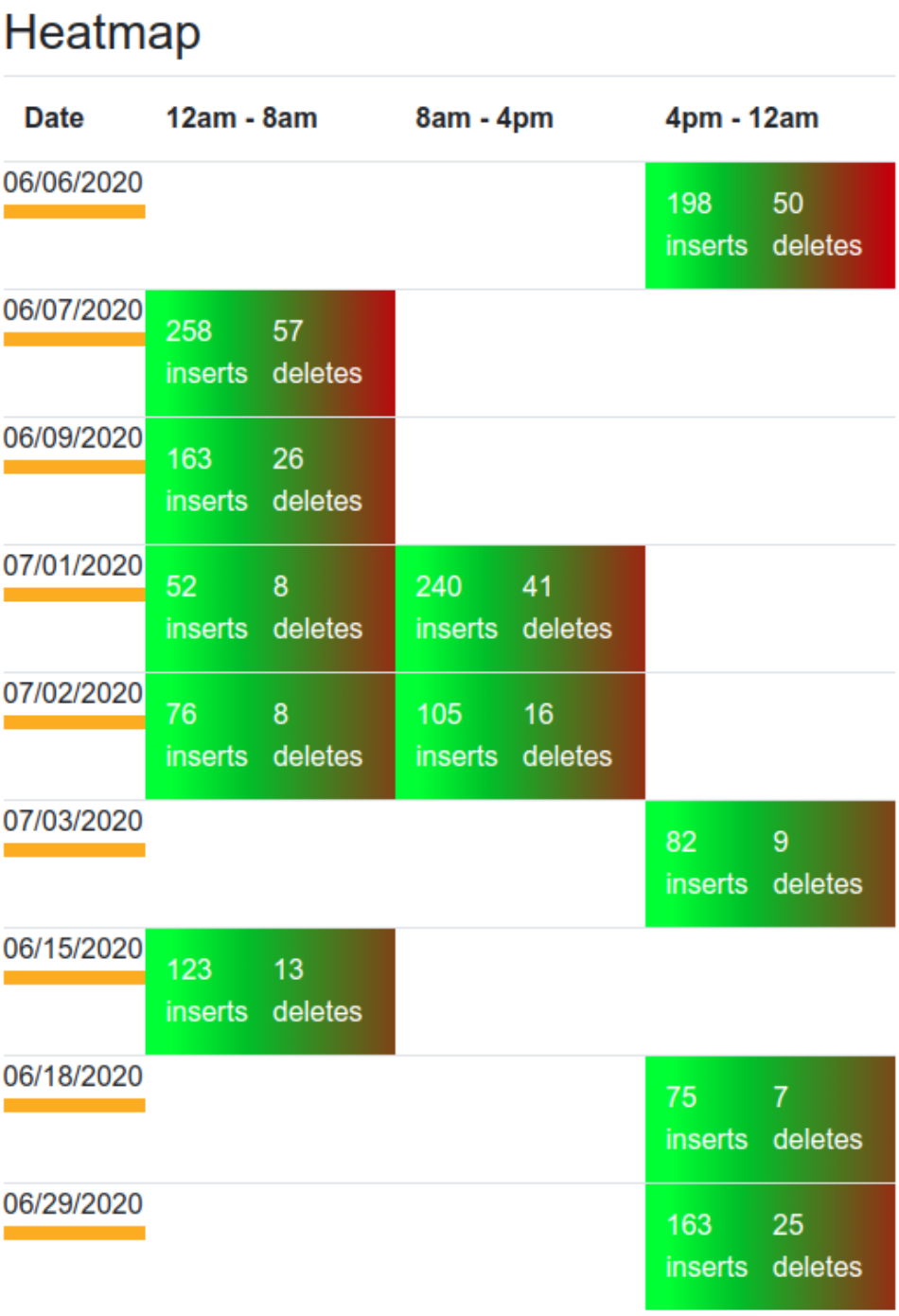


Figure A.9. The heatmap shows insertions and deletions in each work session.

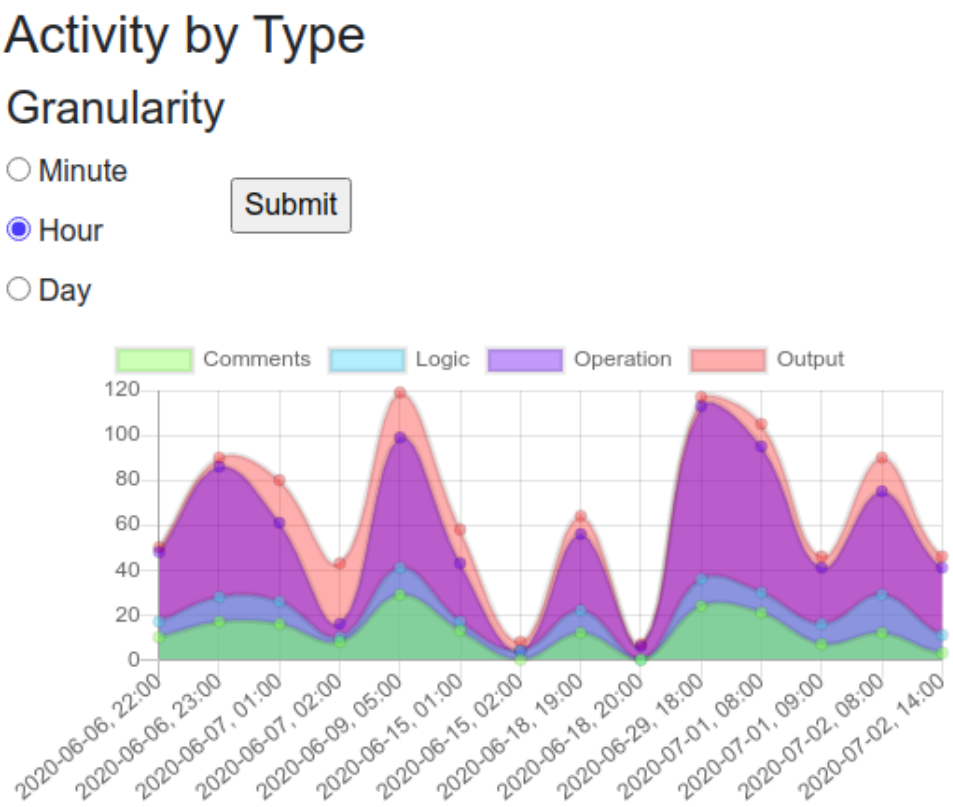


Figure A.10. The activity by type graph shows what type of software development was done at each edit time. The scale can be adjusted per minute, per hour, or per day.

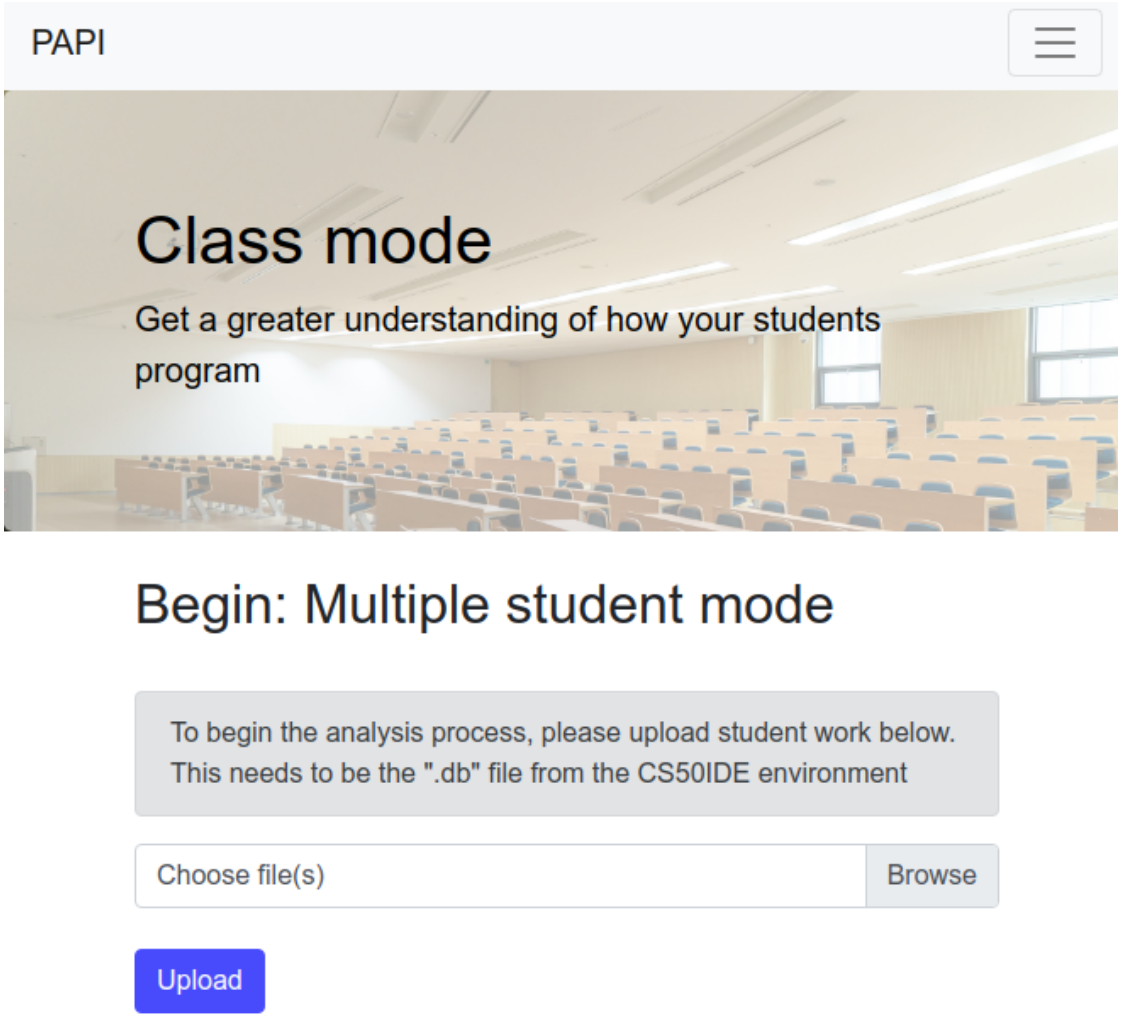


Figure A.11. Multiple files can be selected and uploaded when in multiple student mode.

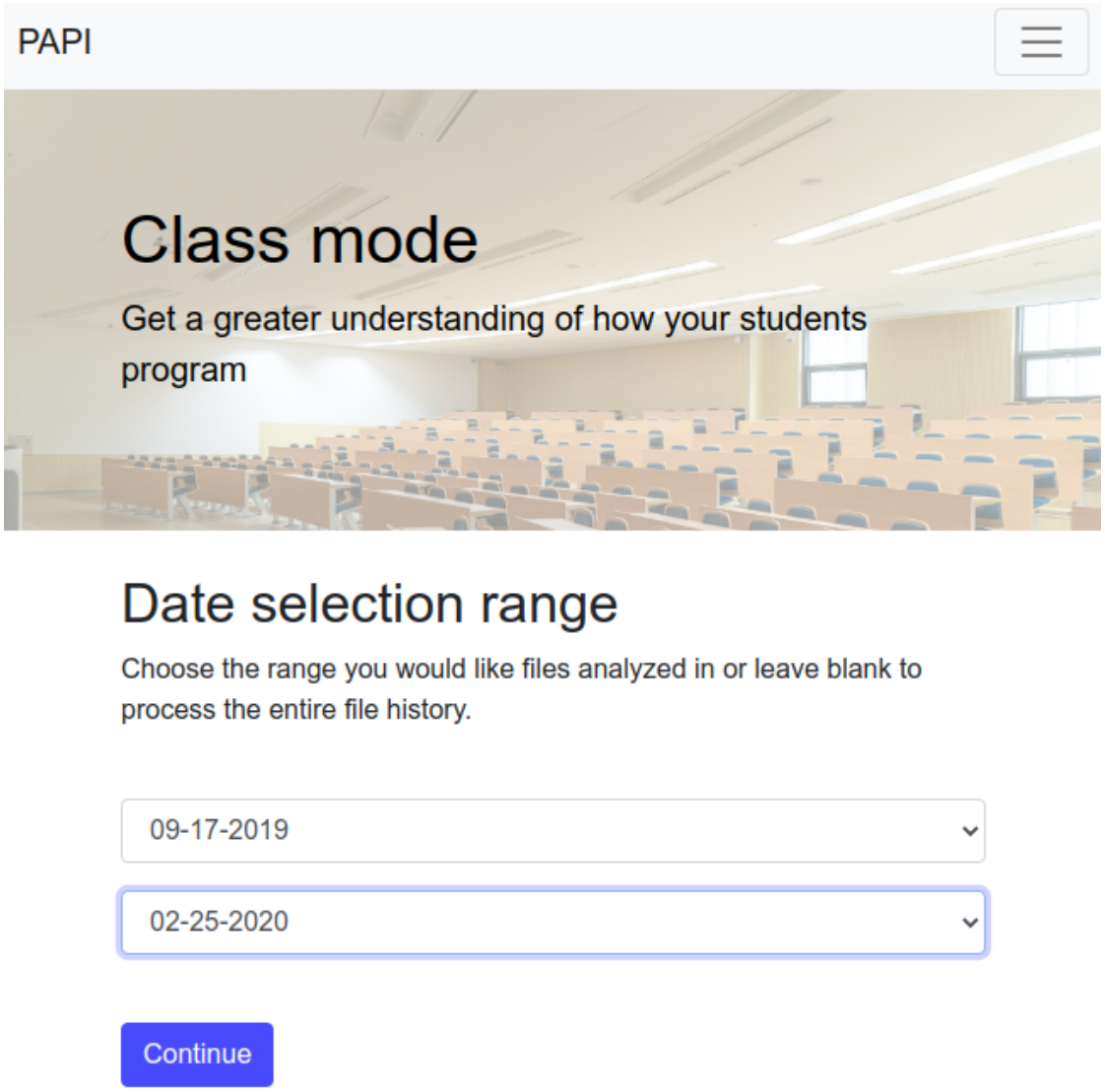


Figure A.12. Multiple student mode allows for date range as the selection criteria.

PAPI

Class Summary

Time Worked	2 days, 5:06:20
Number of Work Sessions	91
Over Number of Days	24
Average Work Session Length	0:35:00.879121

Heatmap

Get insights into when your students are working the most

Date	12am - 8am	8am - 4pm	4pm - 12am
02/03/2020			1023 inserts 312 deletes
02/04/2020			20 inserts 4 deletes
02/05/2020		122 inserts 29 deletes	88 inserts 21 deletes
02/06/2020			49 inserts 13 deletes
01/27/2020			292 inserts 48 deletes
01/29/2020			69 15

Figure A.13. Summary of all uploaded files metrics.

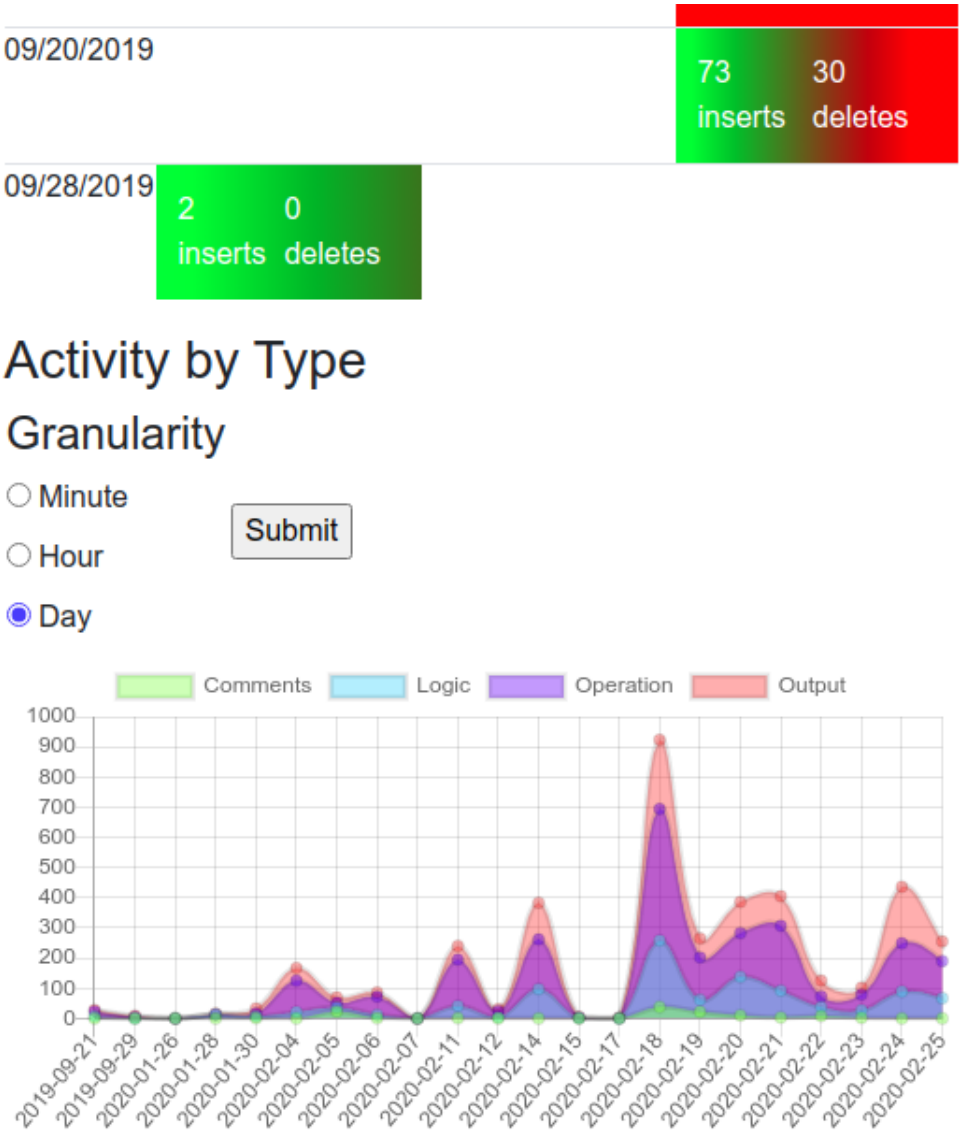


Figure A.14. Summary of heatmap and activity by type graph for all uploaded files.

APPENDIX B

PDF Export File

PAPI Student Feedback Date Generated: 2020-07-10
University of Rhode Island

Name: Sample Student

Time Worked	9:40:45
Number of Work Sessions	54
Over Number of Days	21
Average Work Session Length	0:10:45.277778
First File Creation Date	2020-01-28 19:49:43
Last File Creation Date	2020-02-06 19:29:11
Last Edit Date	2020-02-06 19:29:11
Number of Saves	303
Number of Deletion Characters	6825
Number of Insertion Characters	37075
Number of Comments*	5

Large Text Insertion Detection*

pa1/main_10.cpp

```
  reak;
  case 1:
  std::cout << One << std::endl;
  reak;
  case 1:
  std::cout << One << std::endl;
  reak;
  case 1:
  std::cout << One << std::endl;
  reak;
  case 1:
  std::cout << One << std::endl;
  reak;
  case 1:
  std::cout << One << std::endl;
  reak;
  case 1:
  std::cout << One << std::endl;
```

Figure B.1. Student Feedback Export

PAPI Student Feedback Date Generated: 2020-07-10
University of Rhode Island

Name: Sample Student

```
  reak;
  case 1:
  std::cout << One << std::endl;
  reak;
```

pa1/main_13.cpp

```
  std::cout << Enter numer 1: ;
  std::cin >> num1;
  std::cout << Enter numer 2: ;
  std::cin >> num2;
  std::cout << Enter numer 3: ;
  std::cin >> num3;
  if ()
```

pa1/main_17.cpp

```
  else if (grade == a || grade == A)
  {
  std::cout <<  << std::endl;
  }
  else if (grade == a || grade == A)
  {
  std::cout <<  << std::endl;
  }
  else if (grade == a || grade == A)
  {
  std::cout <<  << std::endl;
  }
  else
  {
  }
```

Figure B.2. Student Feedback Export

PAPI Student Feedback Date Generated: 2020-07-10
University of Rhode Island

Name: Sample Student

pa1/main_18.cpp

```
1
case 2:
std::cout << Saturday << std::endl;
reak;
case 3:
std::cout << Saturday << std::endl;
reak;
case 0:
std::cout << Saturday << std::endl;
reak;
case 0:
std::cout << Saturday << std::endl;
reak;
case 0:
std::cout << Saturday << std::endl;
reak;
```

pa2/main_8.cpp

```
< (n / 2))//top half
{
//front spaces (i from 0 to < n/2)
for(int j = 0; j < i; j++)
{
std::cout <<;
}
//first star
std::cout << *;
//mid spaces
for(int k = 0; k < (2 * (i + 1)); k++)
{
```

Figure B.3. Student Feedback Export

PAPI Student Feedback Date Generated: 2020-07-10
University of Rhode Island

Name: Sample Student

```
  std::cout <<;
  }
  //second star
  std::cout << *;
  /*
  //after spaces
  for(int j = 0; j < (n / 4); j++)
  {
  std::cout <<;
  }
     */
  }
  else if(i  //ottom half (i from n/2 + 1 to n)-1 - ikk
  /*
  */
```

pa2/main_12.cpp

```
  ;
  switch (digit)
  {
  case 0:
  std::cout << 0;
  reak;
  case 1:
  std::cout << 1;
  reak;
  case 2:
  std::cout << 2;
  reak;
  case 3:
  std::cout << 3;
```

Figure B.4. Student Feedback Export

PAPI Student Feedback Date Generated: 2020-07-10
University of Rhode Island

Name: Sample Student

```
  reak;
  case 4:
  std::cout << 4;
  reak;
  case 5:
  std::cout << 5;
  reak;
  case 6:
  std::cout << 6;
  reak;
  case 7:
  std::cout << 7;
  reak;
  case 8:
  std::cout << 8;
  reak;
  case 9:
  std::cout << 9;
  reak;
  case 10:
  std::cout << A;
  reak;
  case 11:
  std::cout << B;
  reak;
  case 12:
  std::cout << C;
  reak;
  case 13:
  std::cout << D;
  reak;
```

Figure B.5. Student Feedback Export

PAPI Student Feedback Date Generated: 2020-07-10
University of Rhode Island

Name: Sample Student

```
  case 14:
  std::cout << E;
  reak;
  case 15:
  std::cout << F;
  reak;
  default:
  std::cout << Error;
  reak;
  }
```

pa2/main_15.cpp

```
  for(int i = n; i < n; i++)
  {
  std::cout << |;
  for(int j = 0; j < (2 * n) - 2; j++)
  {
  std::cout <<;
  }
  std::cout << <>;
  for(int k = 0; k < (i * 4); k++)
  {
  std::cout << .;
  }
  std::cout << <>;
  for(int l = 0; l < (2 * n) - 2; l++)
  {
  std::cout <<;
  }
  std::cout << | << std::endl;
  }
```

Figure B.6. Student Feedback Export

PAPI Student Feedback Date Generated: 2020-07-10
University of Rhode Island

Name: Sample Student

pa3/main_10.cpp

```
std::string void toLower(std::string& str){
for(int i = 0; i < str.length(); i++)
{
if(int(stri) >= 65 && int(stri) <= 90)
{
stri = char(int(stri) + 32);
}
}
return;}
```

Figure B.7. Student Feedback Export

PAPI Student Feedback
University of Rhode Island

Date Generated: 2020-07-10

Name: Sample Student

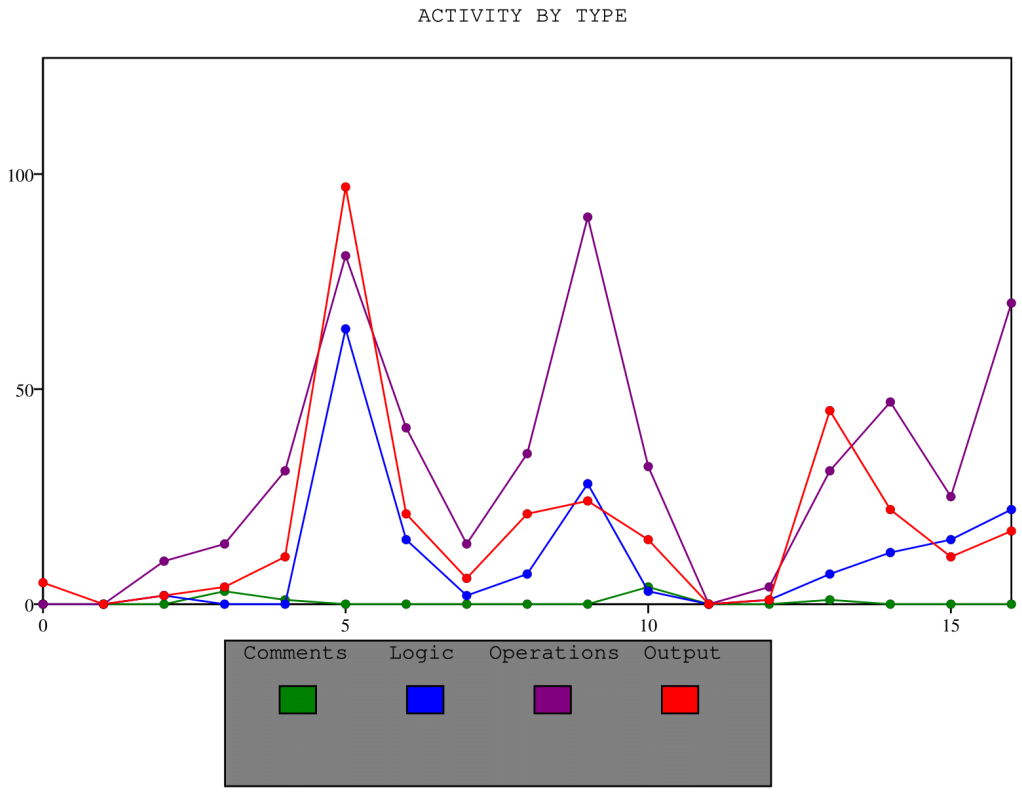


Figure B.8. Student Feedback Export

APPENDIX C

Survey Questions

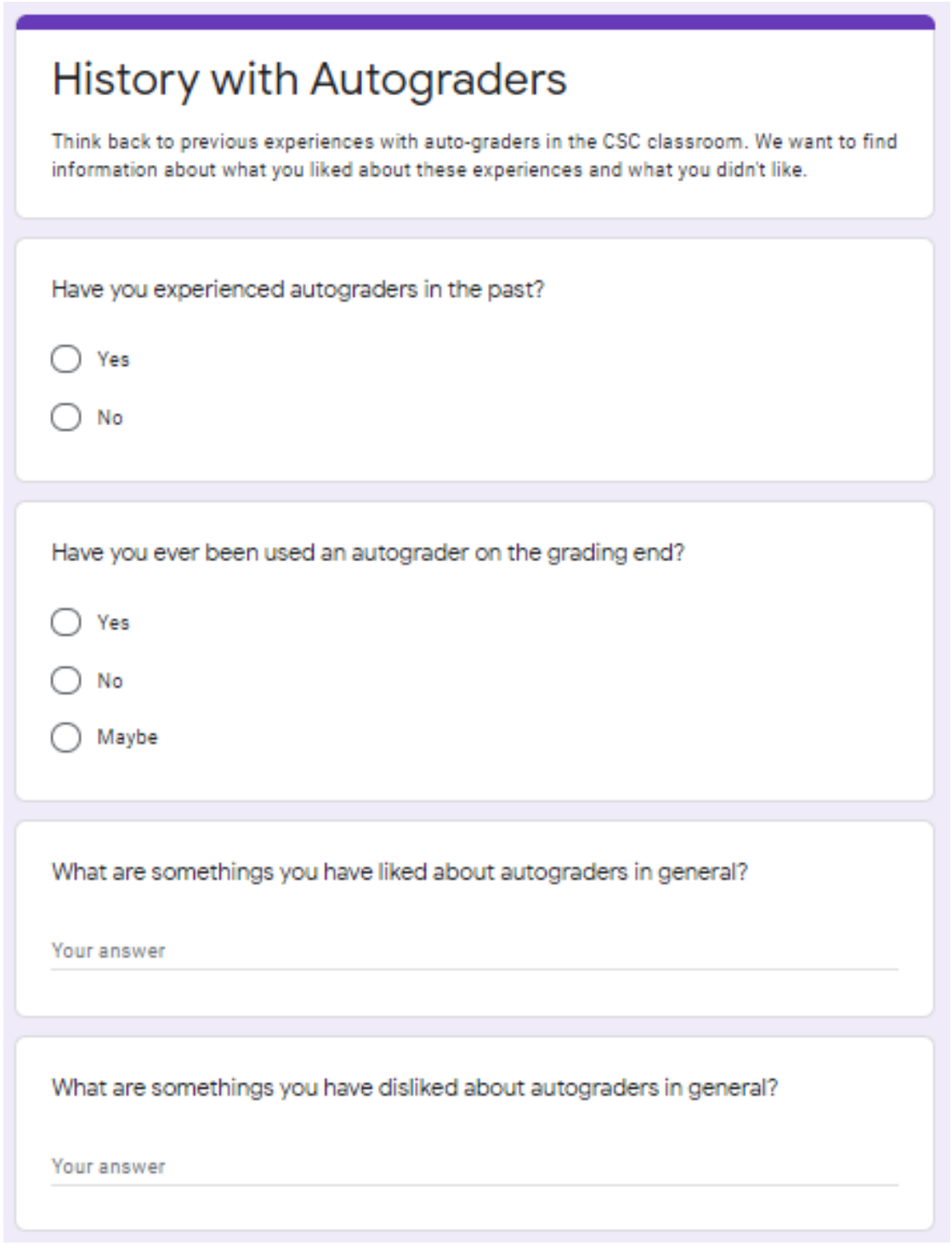


Figure C.1. User Survey

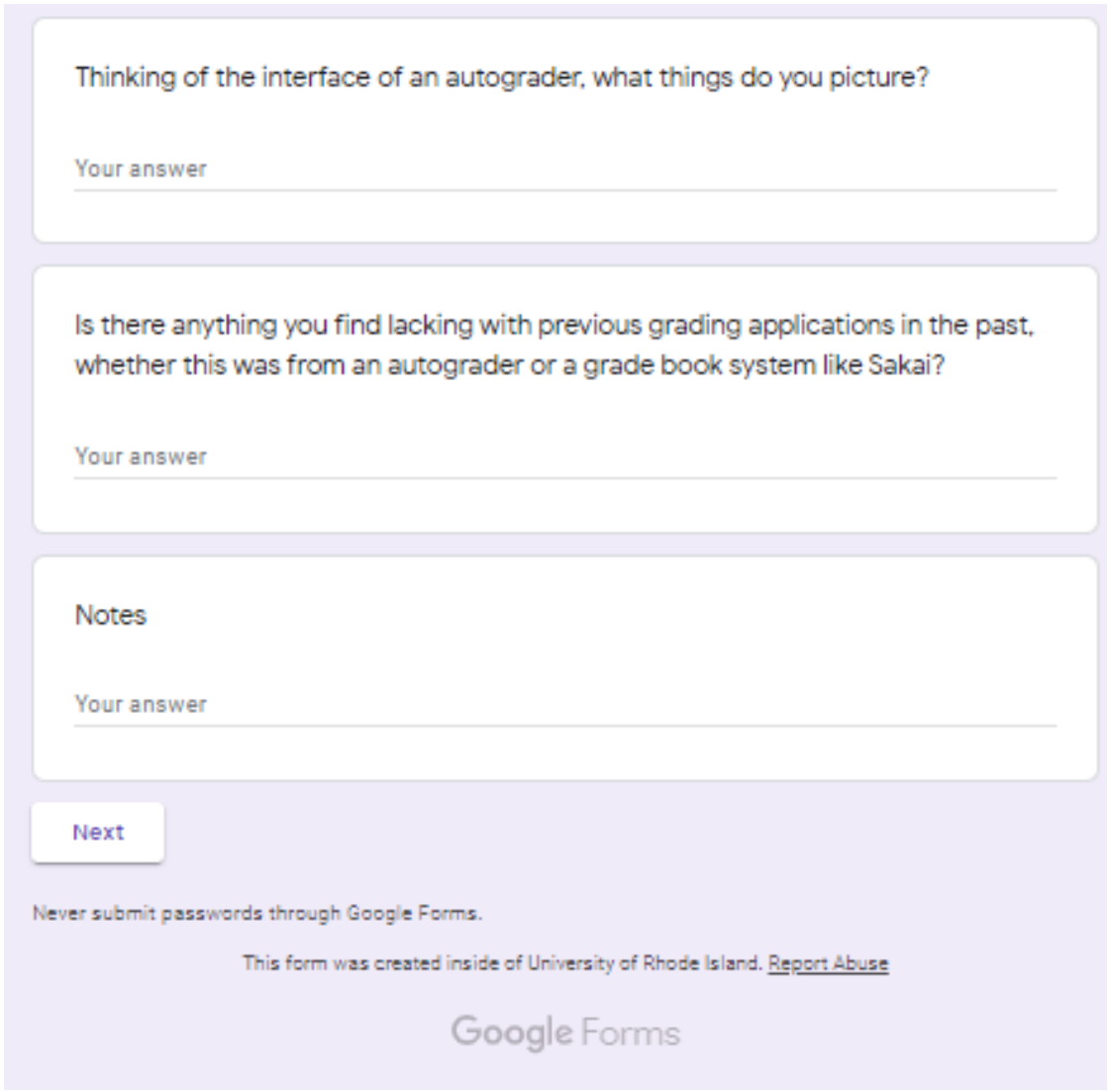


Figure C.2. User Survey

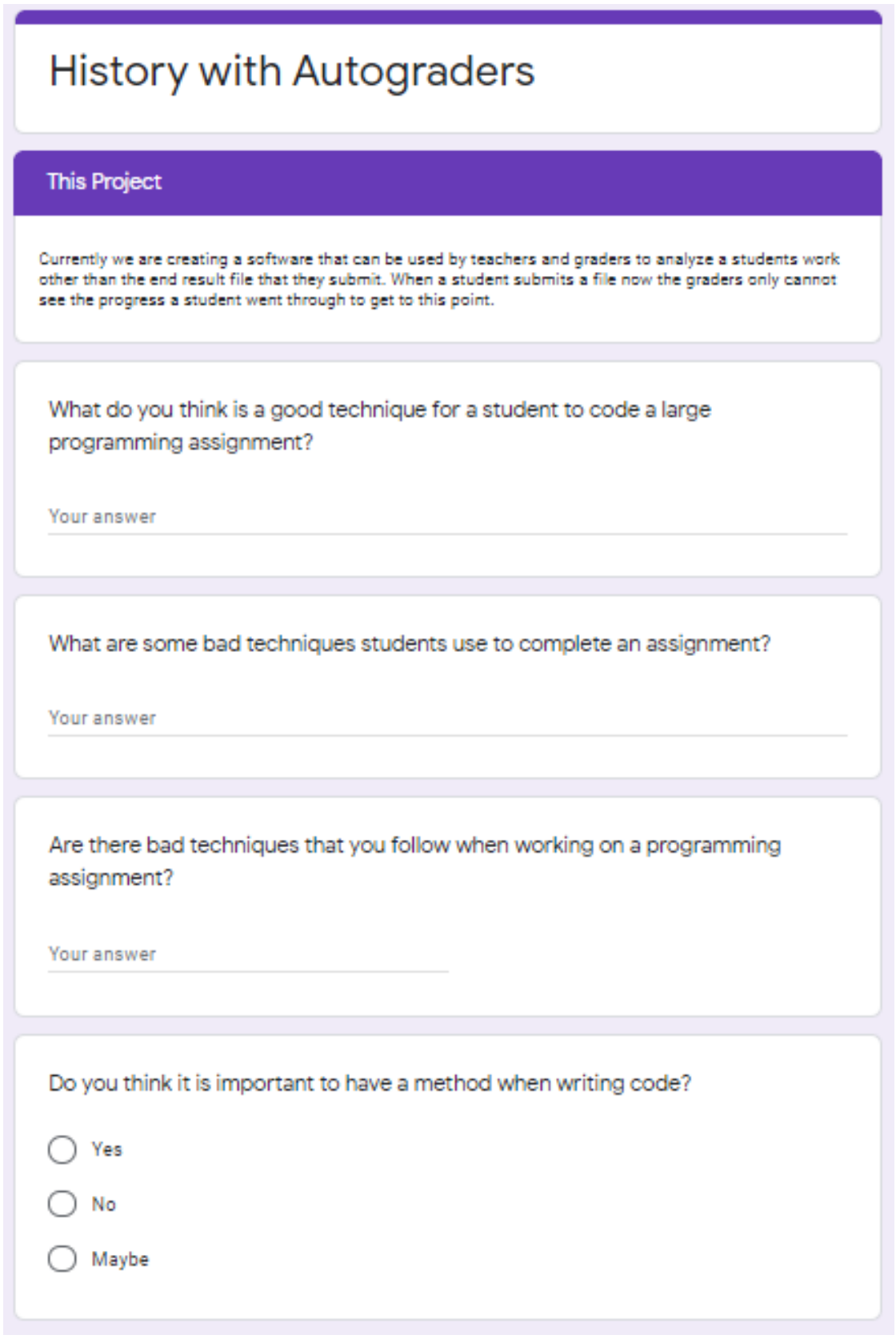


Figure C.3. User Survey

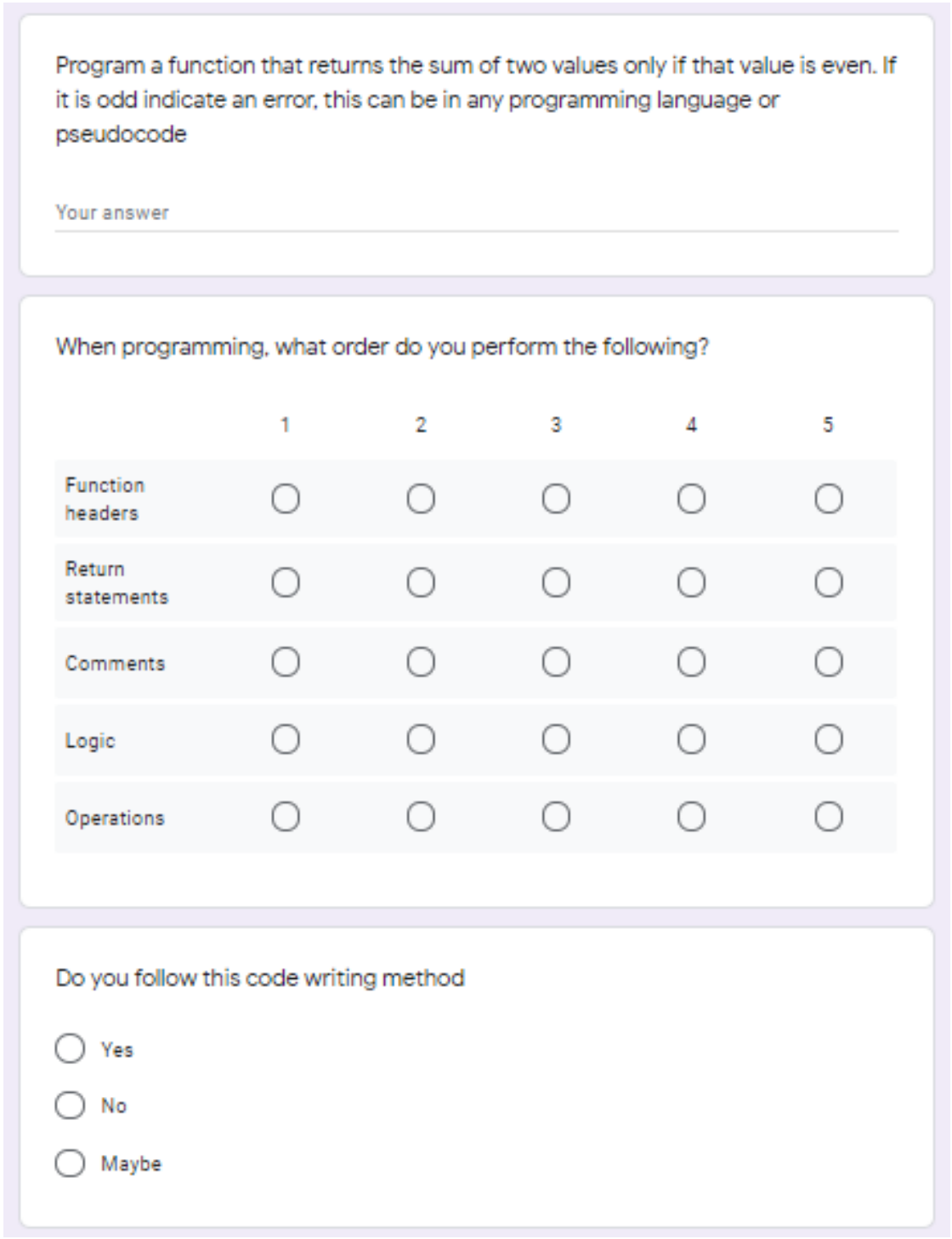


Figure C.4. User Survey

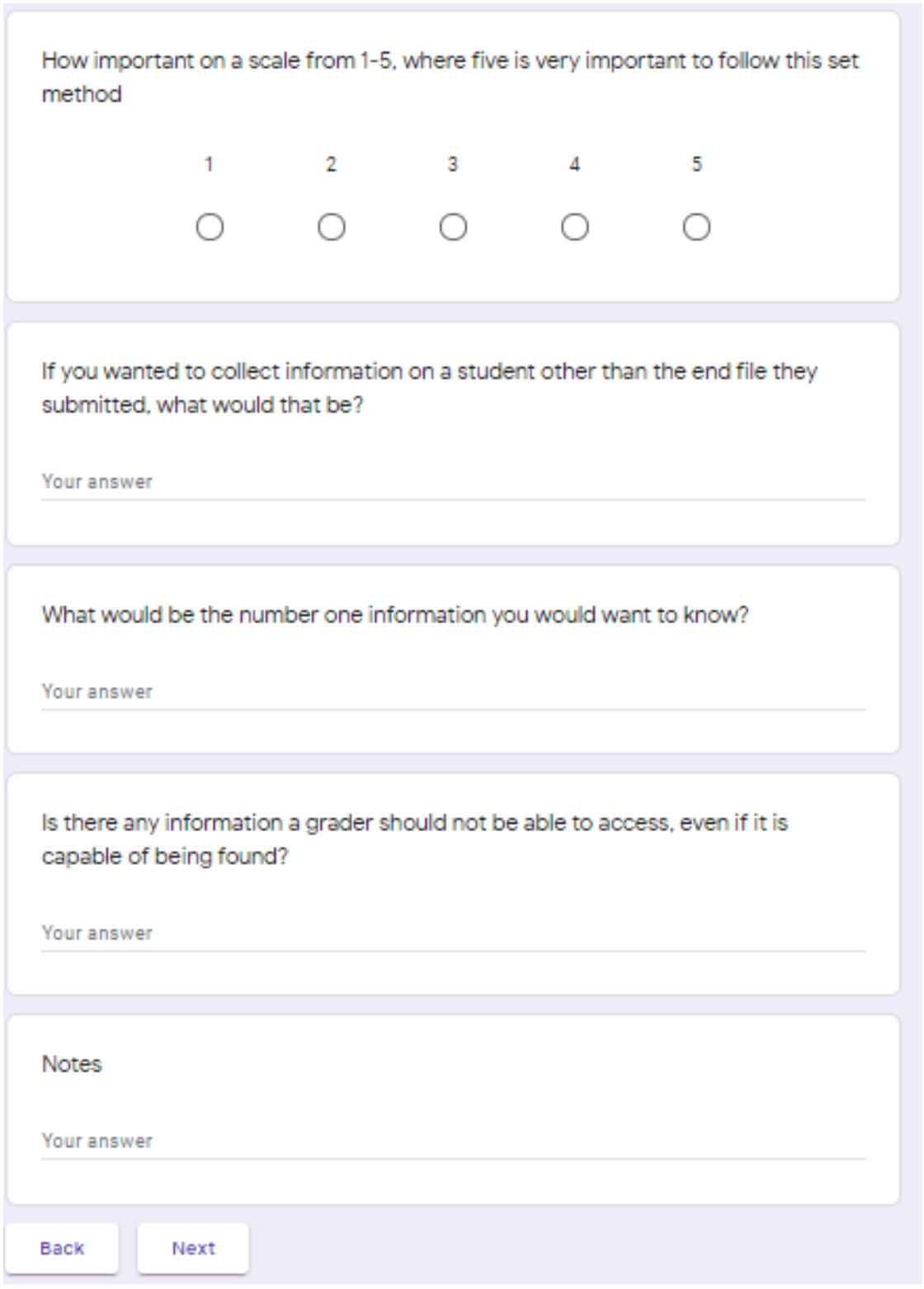


Figure C.5. User Survey

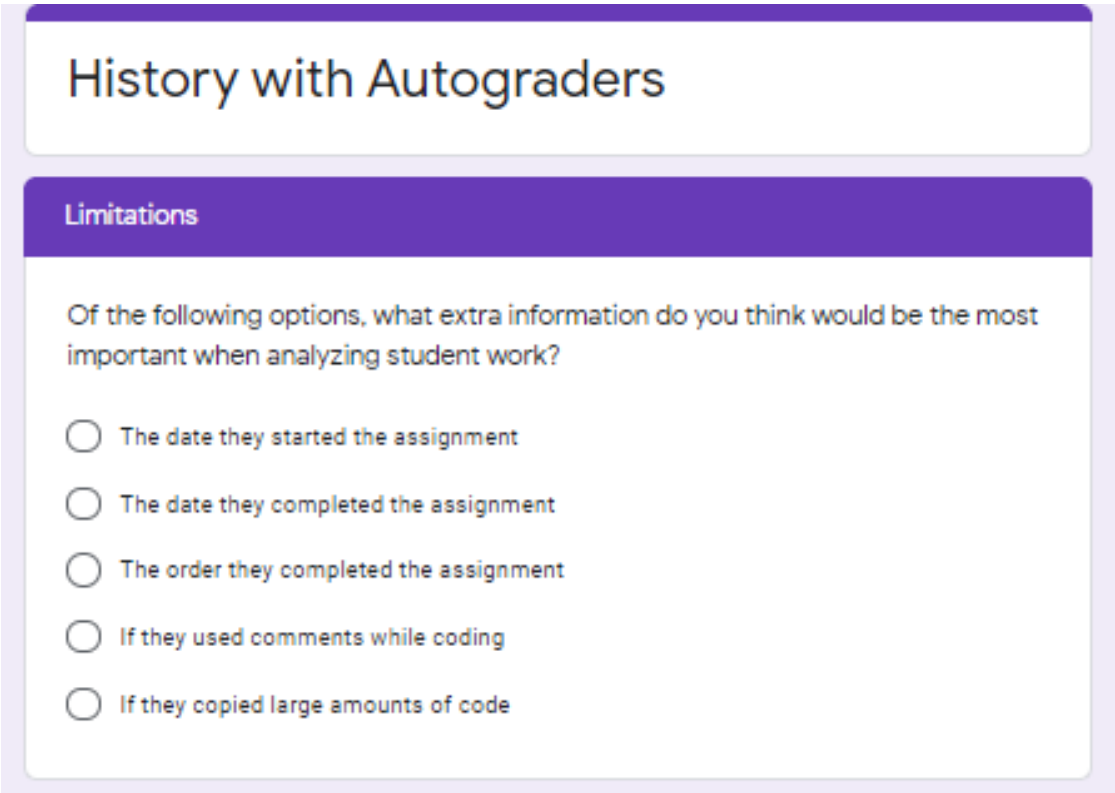


Figure C.6. User Survey

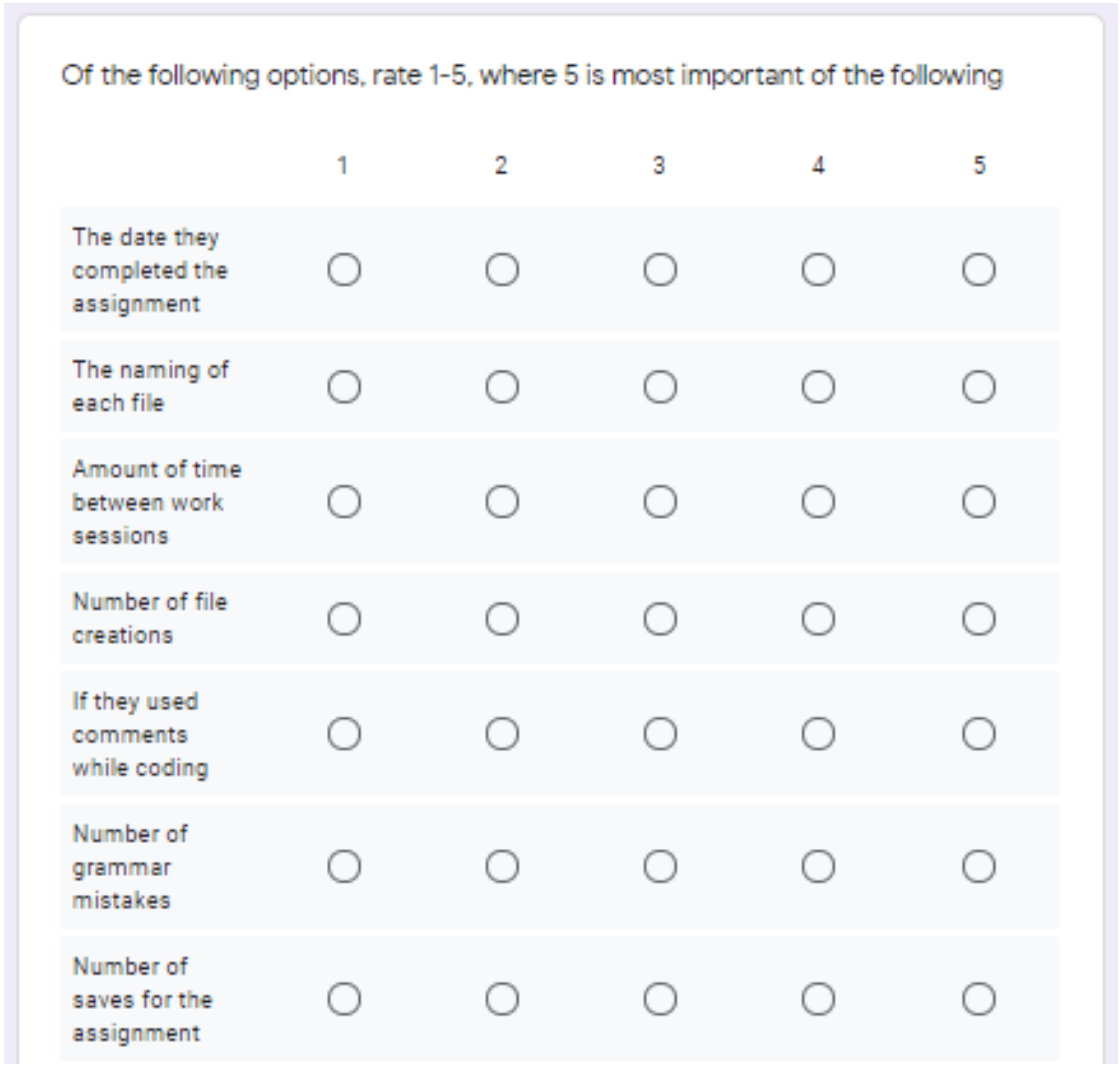


Figure C.7. User Survey

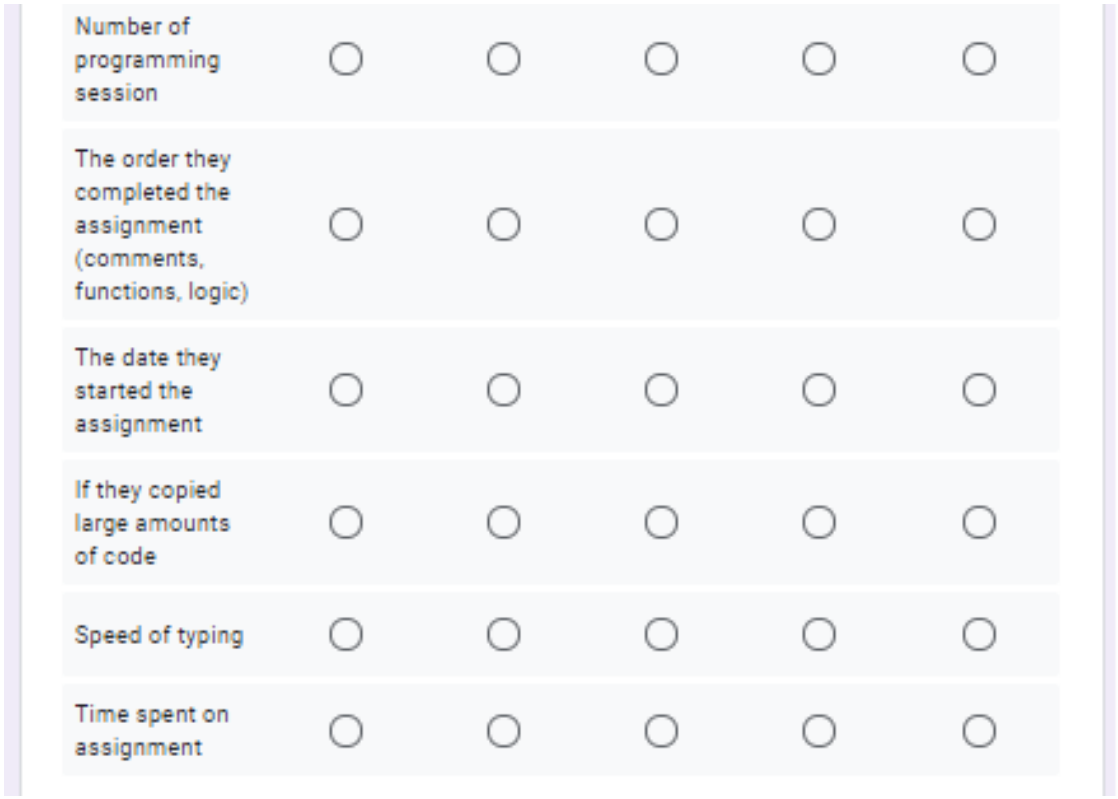


Figure C.8. User Survey

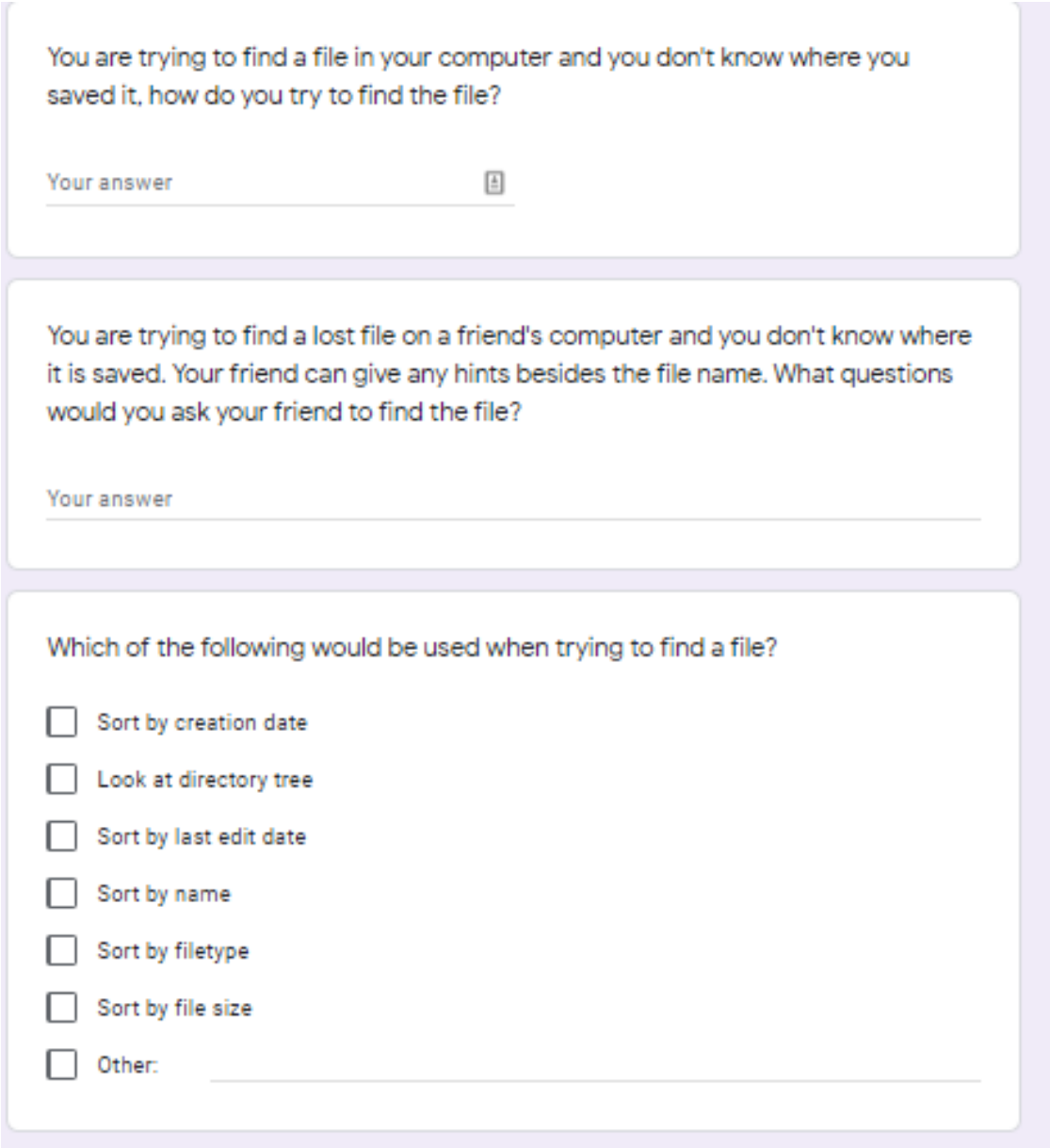


Figure C.9. User Survey

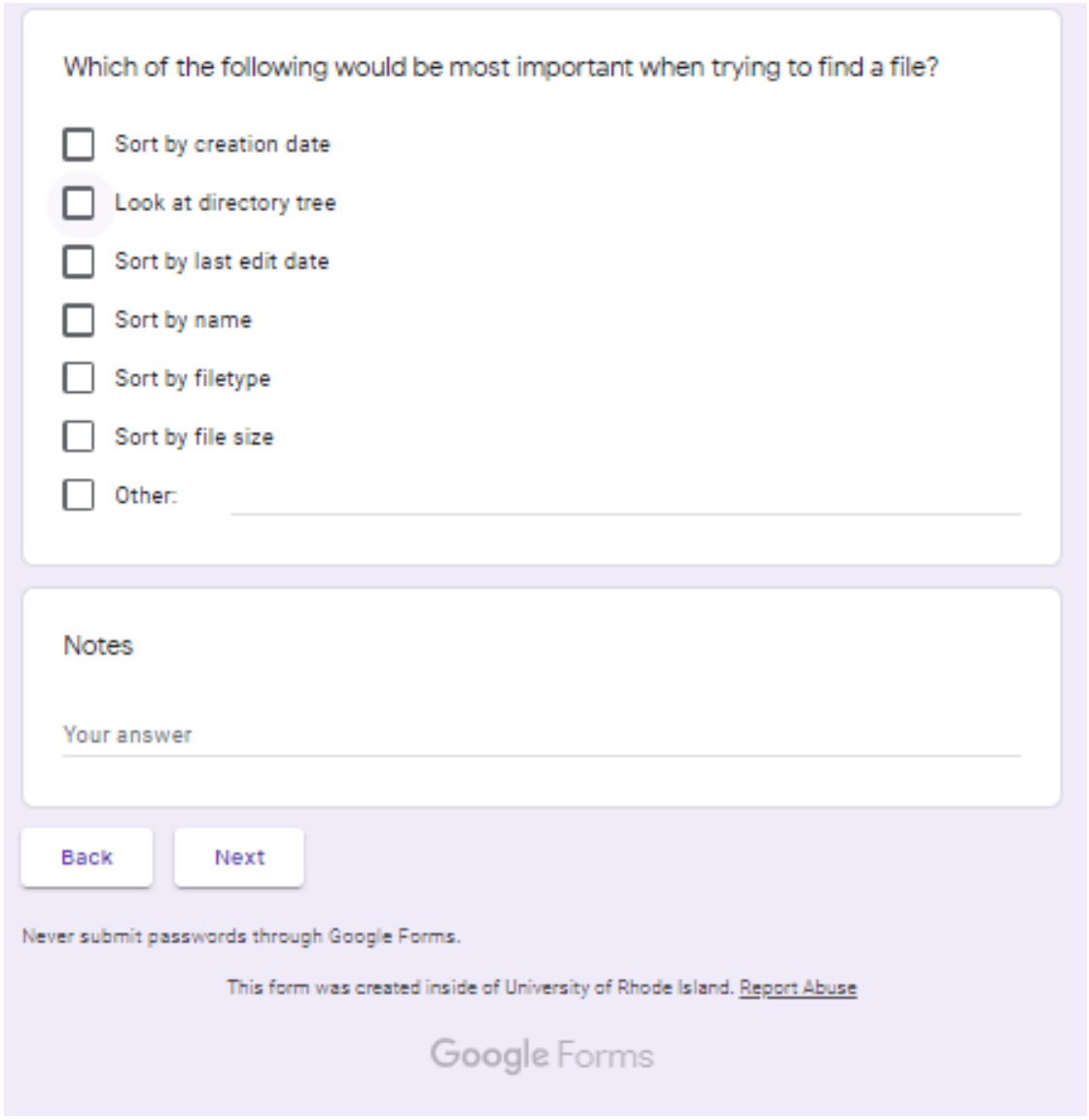


Figure C.10. User Survey

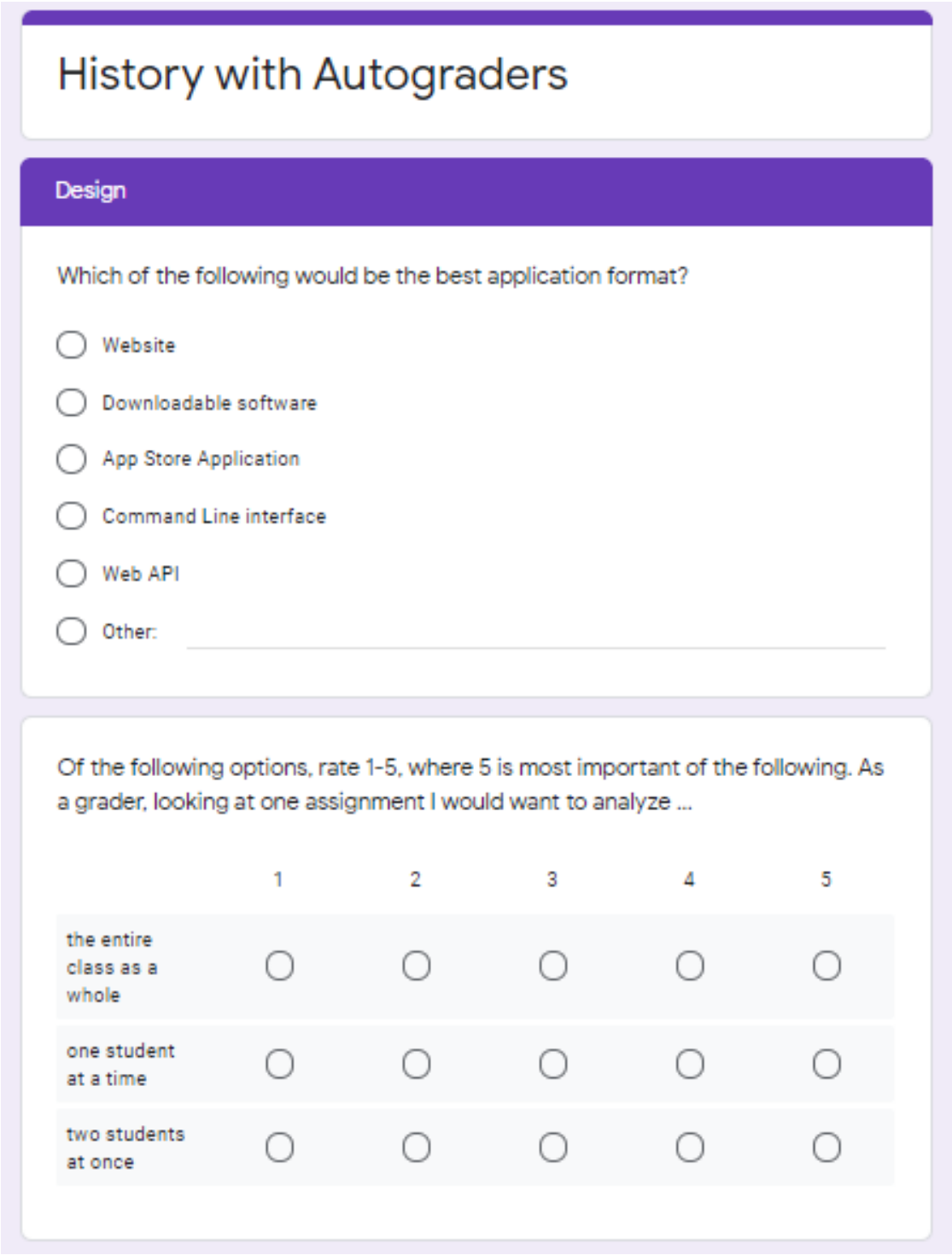


Figure C.11. User Survey

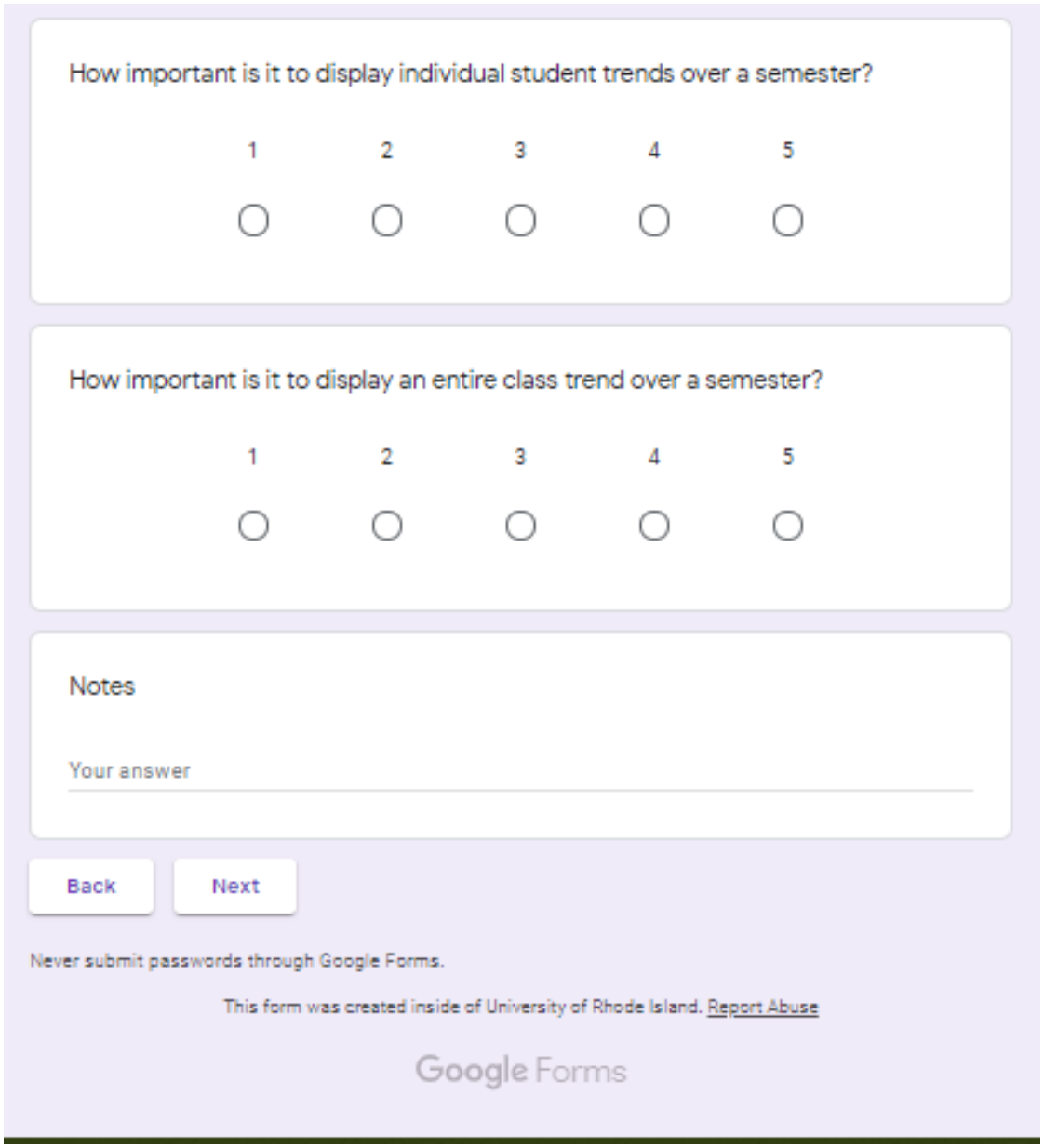


Figure C.12. User Survey

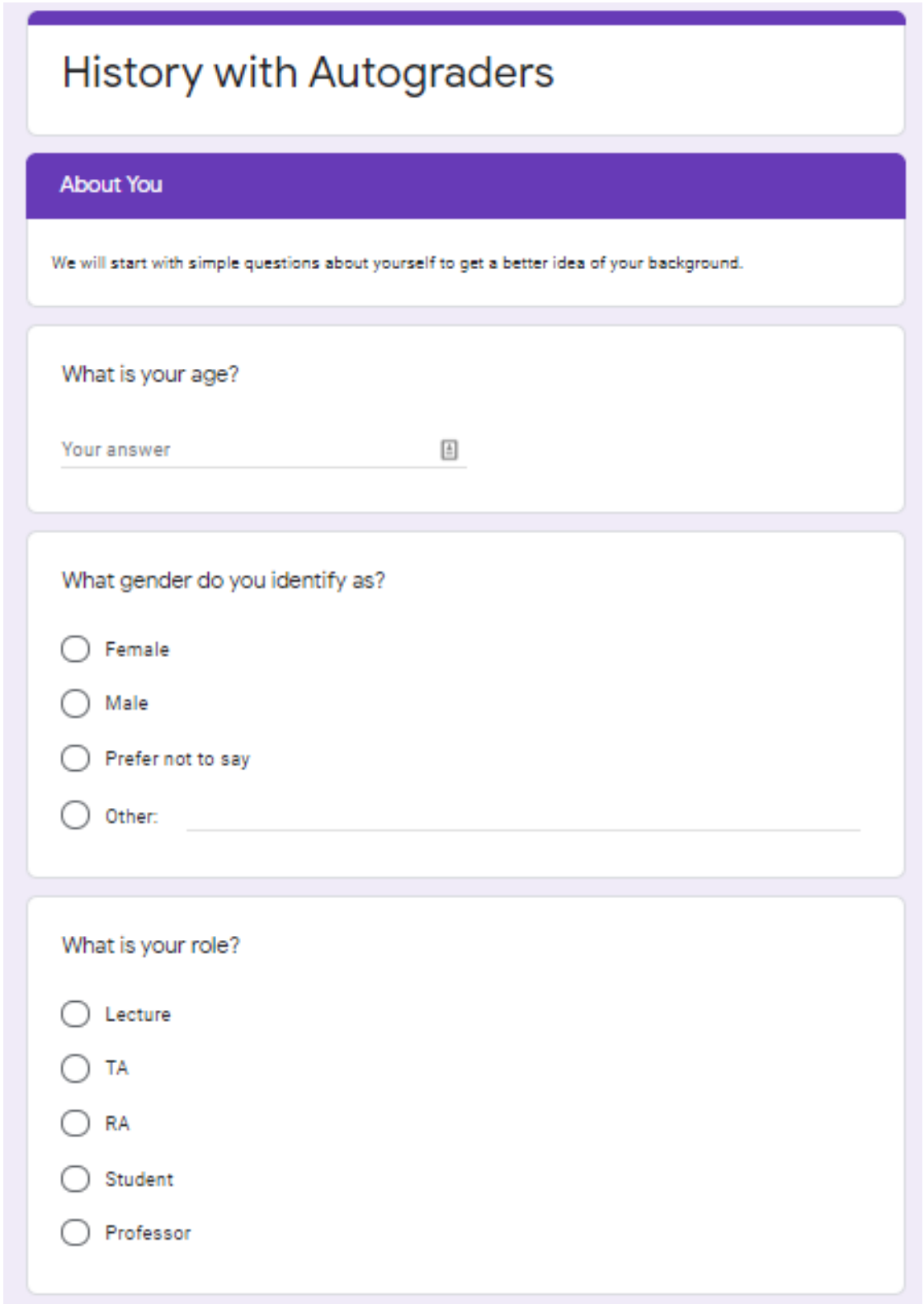


Figure C.13. User Survey

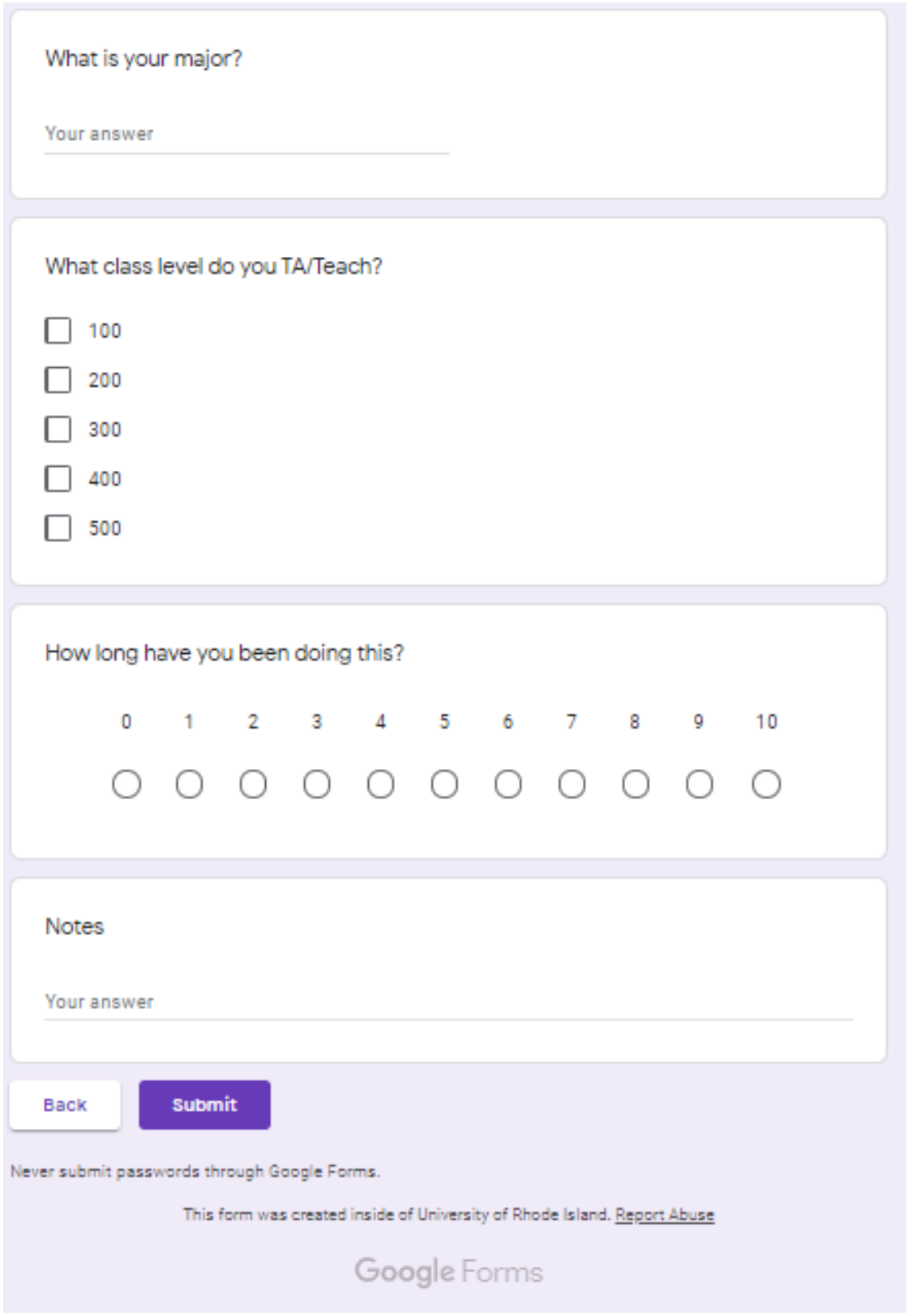


Figure C.14. User Survey

BIBLIOGRAPHY

"Codegrade - deliver engaging feedback on code." [Online]. Available: https://www.codegrade.com/

"Github classroom." [Online]. Available: https://classroom.github.com/

"Gradescope — save time grading." [Online]. Available: https://www.gradescope.com/

"Welcome to the kattis problem archive." [Online]. Available: https://open.kattis.com/

"The strategy on replicate and similar web collections' detecting and clustering," *Computer applications in engineering education*, vol. 20, pp. 221–231, 2012.

Aiken, A., "A system for detecting software similarity." [Online]. Available: https://theory.stanford.edu/~aiken/moss/

Aiken, A., Jul 2020.

Aiken, A. Stanford. "A system for detecting software similarity." 2020. [Online]. Available: https://theory.stanford.edu/~aiken/moss/

Arafat, O. and Riehle, D., "The commenting practice of open source," in *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, ser. OOPSLA '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 857–864. [Online]. Available: https://doi.org/10.1145/1639950.1640047

Bell, J. T., "Extreme programming," 2001.

Brack, F., "Don't copy-paste code. type it out. ?" Jul 2019. [Online]. Available: https://www.freecodecamp.org/news/the-benefits-of-typing-instead-of-copying-54ed734ad849/

Davis, A. The Hoya. "Plagiarism-detection web site must not profit from students' work." October 2006. [Online]. Available: https://thehoya.com/plagiarism-detection-web-site-must-not-profit-from-students-work/

de Oliveira, Clara Amelia; Conte, M. F. R. B. G., ""aspects on teaching/learning with object oriented programming for entry level courses of engineering"." ERIC, 1998. [Online]. Available: https://eric.ed.gov/?id=ED448994

de Souza, S. C. B., Anquetil, N., and de Oliveira, K. M., "A study of the documentation essential to software maintenance," in *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting Designing for Pervasive Information*, ser. SIGDOC '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 68–75. [Online]. Available: https://doi.org/10.1145/1085313.1085331

DePasquale, P. J., Locasto, M. E., Kaczmarczyk, L., and Martinovic, M., "// todo: Help students improve commenting practices," in *2012 Frontiers in Education Conference Proceedings*, 2012, pp. 1–6.

Elena L. Glassman, Lyla Fischer, J. S. and Miller, R. C., "Foobaz: Variable name feedbackfor student code at scale," 2015. [Online]. Available: http://hdl.handle.net/1721.1/116536

Gulwani, S., Radiček, I., and Zuleger, F., "Feedback generation for performance problems in introductory programming assignments," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 41–51. [Online]. Available: https://doi.org/10.1145/2635868.2635912

Handel, M. J., "Trends in job skill demands in oecd countries," no. 143, 2012. [Online]. Available: https://www.oecd-ilibrary.org/content/paper/5k8zk8pcq6td-en

Head, A., Glassman, E., Soares, G., Suzuki, R., Figueredo, L., D'Antoni, L., and Hartmann, B., "Writing reusable code feedback at scale with mixed-initiative program synthesis," in *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, ser. L@S '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 89–98. [Online]. Available: https://doi.org/10.1145/3051457.3051467

Huang, J., Piech, C., Nguyen, A., and Guibas, L., "Syntactic and functional variability of a million code submissions in a machine learning mooc," in *Proceedings of the Workshops at the 16th International Conference on Artificial Intelligence in Education AIED 2013*, 2013.

Hughes, J. L., "Rethinking and updating demographic questions: Guidance to improve descriptions of research samples," *Psi Chi journal of psychological research.*, vol. 21, no. 3, pp. 138–151, 2016.

Igoa, L. B. and Kiewra, K. A., "How do high-achieving students approach web-based, copy and paste note taking? selective pasting and related learning outcomes," *Journal of Advanced Academics*, vol. 18, no. 4, pp. 512–529, 2007. [Online]. Available: https://journals.sagepub.com/doi/full/10.4219/jaa-2007-558

Kaya, M. and Özel, S. A., "Integrating an online compiler and a plagiarism detection tool into the moodle distance education system for easy assessment of programming assignments," *Computer Applications in Engineering Education*, vol. 23, no. 3, pp. 363–373, 2015, iD: $\mathrm{TN}_c di_g ale_i nfotracacademiconefile_A 409574674$.

Kernighan, B. W. and Plauger, P. J., "Programming style: Examples and counterexamples," *ACM Comput. Surv.*, vol. 6, no. 4, p. 303–319, Dec. 1974. [Online]. Available: https://doi.org/10.1145/356635.356641

Malan, D. J., "Cs50ide," Jul 2019. [Online]. Available: https://github.com/cs50/ide

Malan, D. J., Jun 2020.

Monahan, K., Ye, C., Gould, E., Xu, M., Huang, S., Spickard, A., Rosenbloom, S. T., Coco, J., Fabbri, D., and Miller, B., "Copy-and-paste in medical student notes: Extent, temporal trends, and relationship to scholastic performance," *Applied Clinical Informatics*, vol. 10, no. 3, pp. 479–486, 2019. [Online]. Available: http://dx.doi.org/10.1055/s-0039-1692402

Nguyen, A., Piech, C., Huang, J., and Guibas, L., "Codewebs: Scalable homework search for massive open online programming courses," in *Proceedings of the 23rd International World Wide Web Conference (WWW 2014)*, Seoul, Korea, 2014.

Oman, P. W. and Cook, C. R., "Programming style authorship analysis," in *Proceedings of the 17th Conference on ACM Annual Computer Science Conference*, ser. CSC '89. New York, NY, USA: Association for Computing Machinery, 1989, p. 320–326. [Online]. Available: https://doi.org/10.1145/75427.75469

Oman, P. W. and Cook, C. R., "A taxonomy for programming style," in *Proceedings of the 1990 ACM Annual Conference on Cooperation*, ser. CSC '90. New York, NY, USA: Association for Computing Machinery, 1990, p. 244–250. [Online]. Available: https://doi.org/10.1145/100348.100385

Palomba, F., Bavota, G., Di Penta, M., Oliveto, R., De Lucia, A., and Poshyvanyk, D., "Detecting bad smells in source code using change history information," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'13. IEEE Press, 2013, p. 268–278. [Online]. Available: https://doi-org.uri.idm.oclc.org/10.1109/ASE.2013.6693086

Prechelt, L., Malpohl, G., and Philippsen, M., "Finding plagiarisms among a set of programs with jplag," *Journal of Universal Computer Science*, vol. 8, no. 11, pp. 1016–1038, 2002.

Price, B. and Petre, M., "Teaching programming through paperless assignments: An empirical evaluation of instructor feedback," in *Proceedings of the 2nd Conference on Integrating Technology into Computer Science Education*, ser. ITiCSE '97.

New York, NY, USA: Association for Computing Machinery, 1997, p. 94–99. [Online]. Available: https://doi.org/10.1145/268819.268849

Rivers, K. and Koedinger, K. R., "Automatic generation of programming feedback: A data-driven approach," *AAIED 2013 Workshops Proceedings*, vol. 9, pp. 50–59, 2013.

Rubiano, S. M. M., López-Cruz, O., and Soto, E. G., "Teaching computer programming: Practices, difficulties and opportunities," in *2015 IEEE Frontiers in Education Conference (FIE)*, 2015, pp. 1–9.

Rudge, M., "Why you should never copy and paste code from the web," Mar 2020. [Online]. Available: https://levelup.gitconnected.com/why-you-should-never-copy-and-paste-code-from-the-web-41584544036

Schleimer, S., Wilkerson, D. S., and Aiken, A., "Winnowing: Local algorithms for document fingerprinting," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 76–85. [Online]. Available: https://doi.org/10.1145/872757.872770

Singh, R., Gulwani, S., and Solar-Lezama, A., "Automated feedback generation for introductory programming assignments," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 15–26. [Online]. Available: https://doi.org/10.1145/2491956.2462195

Stapel, K., Lübke, D., and Knauss, E., "Best practices in extreme programming course design," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 769–776. [Online]. Available: https://doi.org/10.1145/1368088.1368197

Thimbleby, H., "Better programming," 2004.

Toolan, T. M. and Tufts, D. W., "Detection and estimation in non-stationary environments," in *Proceedings IEEE Asilomar Conference on Signals, Systems & Computers*, Nov. 2003, pp. 797–801.

Turabian, K. L., *A Manual for Writers of Term Papers, Theses, and Dissertations, 6th. edn.* Chicago, Illinois, United States of America: University of Chicago Press, 1987.

University of Rhode Island. "A guide to producing your thesis with latex." June 2006. [Online]. Available: http://www.ele.uri.edu/info/thesis/guide

Vihavainen, A., Paksula, M., and Luukkainen, M., "Extreme apprenticeship method in teaching programming for beginners," in *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 93–98. [Online]. Available: https://doi.org/10.1145/1953163.1953196

Wise, M. J., "Yap3: Improved detection of similarities in computer program and other texts," in *Proceedings of the Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 130–134. [Online]. Available: https://doi.org/10.1145/236452.236525

Wong, E., Yang, J., and Tan, L., "Autocomment: Mining question and answer sites for automatic comment generation," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'13. IEEE Press, 2013, p. 562–567. [Online]. Available: https://doi-org.uri.idm.oclc.org/10.1109/ASE.2013.6693113

Yuen, C. K., "The programmer as navigator: A discourse on program structure," *SIGPLAN Not.*, vol. 18, no. 9, p. 70–78, Sept. 1983. [Online]. Available: https://doi.org/10.1145/988227.988236